Final Project Report

Version 1.1 – 2023.12.04

Created 2023.12.04

Project Name: Creating an Accessible RPG
Class: CSC207: Introduction to Software Design
Professor: Sonya Allin
TA: Sahil Gupta
Group Number: 37

Creators:
Madhav Ajayamohan
Panth Barola
Ndaru Aryya Bhramastra
Willem Ethan Tio

GitLab Repository:
https://mcsscm.utm.utoronto.ca/csc207_20239/group_37

Section 1: Introduction

Our main motivation for this project is our love for games. All of us have been playing games since we were kids, and this is the first chance we have to actually ‘create’ a game, with the advanced skills we have learned so far.

We will be building off of the Adventure Game that was created in assignment two, and improving upon it. Firstly, we will be adding more accessibility features such as the ability to change brightness, a high contrast, the ability to change text size and more so that players with visual and/or auditory impairments can enjoy the game.

We also plan to improve upon how the user’s enjoy this game, by adding several new features, such as background music, new Troll sub-games, rewarding player achievements and a new power up system.

Section 2: Sprint One Process Documentation

Section 2.1 Sprint Overview

The first sprint started on November 10, 2023 and ended on November 17, 2023. During this time period our goals are to:

- Implement the state design pattern to create a main room traversal screen, an inventory screen and a settings screen,
- Implement a *enlargeText* and *minimizeText* method to increase and decrease text size of the text in the game,
- Implement the memento design pattern to give the game an ‘autosave function,’ where the game automatically after the game window is closed,
- Implement the singleton design pattern to play music in the background of the game,
- Implement the decorator design pattern on the Player class to be able to give the player buffs (increased attack or defense) or debuffs (decreased attack or defense).

Section 2.2 Stories Selected for Sprint One

User Story	Assigned to	Priority	Changes in Implementation
4.2 Autosave	Ndaru Bhramastra	1	Changes reflected in the updated UML diagram below.
2.5 Item Effects	Willem Ethan Tio	1	Changes reflected in UML, decorators were created to handle effects.
1.6 Different Screens	Madhav Ajayamohan	1	Changes reflected in the updated UML diagram below.
1.7 Settings Menu	Madhav Ajayamohan	1	Changes reflected in updated UML diagram below.
1.3 Change Text	Madhav Ajayamohan	1	*enlargeText* and *minimizeText* will not be called with the key commands 'Command/Control +' and 'Command/Control -'. Furthermore, the *enlargeText* is now implemented as *addEnlargeEvent* and *minimizeEvent* is now implemented as *addMinimizeEvent*. The text size changed by changing the value of a static variable *textsize* in the *GridState* class. This variable is used to determine the text size of all text in each *GridState object*.
2.4 Room Effects	Willem Ethan Tio	2	Changes reflected in UML, decorators were created to handle effects.
4.3 Background Music	Panth Barola	3	Changes reflected in updated UML diagram below

Section 2.3 Team Capacity

The goal is to complete all user stories (except user story 2.4 Room Effect and user story 2.5 Item Effects) by November 16, 2023. Thus, it will be possible to upload all the user stories on GitLab and merge all the code generated in sprint one by November 17, 2023.

User stories 2.4 Room Effect and 2.5 Item Effects will be finished during sprint two.

However, a completed version of some effects will be pushed and merged to the ‘devBranch’ in the first sprint. The User stories 2.4 Room Effect and 2.5 Item Effects will be handled by the same decorator design. This first sprint includes the conversion of player to an abstract class, the completion of the concrete player class, the base decorator, and each of the corresponding buff and debuff wrapper classes.

Section 2.4 Collaborators

Collaborators	Responsibilities
Madhav Ajayamohan	User Story 1.6 Different Screens User Story 1.7 Settings Menu User Story 1.3 Change Text
Panth Barola	User Story 4.3 Background Music
Ndaru Bhramastra	User Story 4.2 Autosave
Willem Ethan Tio	User Story 2.4 Room Effects User Story 2.5 Item Effects Merging ‘devBranch’ with the ‘main’ branch to signify all code for sprint one has been completed, and all components work together.

Section 2.5 Completed and Incomplete Tasks

The completed tasks include:

- User Story 1.6 Different Screens
- User Story 1.7 Settings Menu
- User Story 1.3 Change Text
- User Story 4.2 Autosave

The incomplete tasks include:

- User Story 2.4 Room Effects
- User Story 2.5 Item Effects
- User Story 4.3 Background Music

Section 2.6 Product Backlog

User Story	Assigned to	Priority
1.1 Brightness	Ndaru Bhramastra	1
1.2 High Contrast	Panth Barola	1
1.4 Replay Sound	Ndaru Bhramastra	1
1.5 Audio Help	Panth Barola	1
2.3 Health System	Willem Ethan Tio	1
3.3 Achievement List	Panth Barola	1
2.6 Trolls	Madhav Ajayamohan	1
2.5 Item Effects	Willem Ethan Tio	1
4.4 Game Design	Panth Barola	1
2.1 Hover	Ndaru Bhramastra	2
2.4 Room Effects	Willem Ethan Tio	2
3.1 Text to Audio	Madhav Ajayamohan	3
3.2 Command to Audio	Madhav Ajayamohan	3
2.2 Background Color	Panth Barola	3
4.3 Background Music	Panth Barola	3
4.1 Hints	Willem Ethan Tio	4

Section 2.7 Sprint One Code Reviews

Story Reviewed	Name of Reviewer	Pull Request Link
ID 4.2: Autosave	Willem Ethan Tio	https://mcsscm.utm.utoronto.ca/csc207_20239/group_37/-/merge_requests/4
ID 1.6: Different Screens ID 1.7: Settings Pane ID 1.3: Change Text	Panth Barola	https://mcsscm.utm.utoronto.ca/csc207_20239/group_37/-/merge_requests/2
ID 2.4 Room Effects ID 2.5 Item Effects	Madhav Ajayamohan	https://mcsscm.utm.utoronto.ca/csc207_20239/group_37/-/merge_requests/7

Section 2.8 Sprint One Retrospective

In our first sprint retrospective, on November 16, all members were in attendance. The main focus of this meeting was to check in on each other's progress and try merging branches together on GitLab for the first time together. The branch 'UserStoryDifferentScreens' was merged into the 'devBranch' during the meeting. However, there were some delays in understanding how to merge branches, and all branches could only be merged together by Tuesday of the next week.

During this meeting, we reflected on how the first sprint went. Overall, we all made good progress and nearly completed all assigned responsibilities during this time. The only user story that was not complete– user story 4.3 Background Music– because it was more difficult to implement than initially thought. Therefore, it has been moved to be completed by the next sprint.

The meeting itself was productive and enjoyable: there was a congenial atmosphere that did not obstruct our work, and elevated the quality of our code. The only practice we would like to improve upon is showing up to the meeting on time. During this meeting, members arrived at various times, which made it hard to collaborate. This is a practice we would like to improve upon in the future.

Section 3: Sprint Two Process Documentation

Section 3.1 Sprint Overview

The second sprint started on November 17, 2023 and ended on November 24, 2023. During this time period our goals are to:

- Implement the *getBrightness* method to change the brightness of the game screen,
- Implementing code to process how a Player gets a buff or debuff via items or rooms,
- Implement the singleton design pattern to play music in the background of the game,
- Implement the abstract GameTroll class, and implement three concrete subclasses of GameTroll to create a troll game.

Section 3.2 Stories Selected for Sprint Two

User Story	Assigned to	Priority	Changes in Implementation
1.1 Brightness	Ndaru Bhramastra	1	The method implemented is *addSliderEvent* instead of *getBrightness*. The brightness is changed by changing the static integer variable *brightness* in the abstract *GridState* class. The brightness in every *GridState* object is set using the *brightness* variable.
2.5 Item Effects	Willem Ethan Tio	1	Worked on modifying AdventureGame and TraversalState to handle items that increased the player's attack or defense power. In particular, movePlayer, interpretAction, and submitEvent were modified.
2.6 Trolls	Madhav Ajayamohan	1	Changes reflected in the updated UML diagram below.

2.4 Room Effects	Willem Ethan Tio	2	Worked on modifying AdventureGame and TraversalState to handle room debuffs that would decrease the player's attack or defense power. In particular, movePlayer, interpretAction, and submitEvent were modified.
4.3 Background Music	Panth Barola	3	Changes reflected in updated UML diagram below

Section 3.3 Team Capacity

The goal is to complete all user stories by November 17, 2023. Thus, it will be possible to upload all the user stories on GitLab and merge all the code generated in sprint two by November 24, 2023.

Section 3.4 Collaborators

Collaborators	Responsibilities
Madhav Ajayamohan	User Story 2.6 Trolls
Panth Barola	User Story 4.3 Background Music
Ndaru Bhramastra	User Story 1.1 Brightness
Willem Ethan Tio	User Story 2.4 Room Effects User Story 2.5 Item Effects Merging 'devBranch' with the 'main' branch to signify all code for sprint two has been completed, and all components work together.

Section 3.5 Completed and Incomplete User Stories

There are no complete user stories. The incomplete tasks include:

- User Story 1.1 Brightness
- User Story 2.4 Room Effects
- User Story 2.5 Item Effects
- User Story 2.6 Trolls
- User Story 4.3 Background Music
- To create a new game file for our game.

Section 3.6 Product Backlog

User Story	Assigned to	Priority
1.1 Brightness	Ndaru Bhramastra	1
1.2 High Contrast	Panth Barola	1
1.4 Replay Sound	Ndaru Bhramastra	1
2.3 Health System	Willem Ethan Tio	1
3.3 Achievement List	Panth Barola	1
2.6 Trolls	Madhav Ajayamohan	1
2.5 Item Effects	Willem Ethan Tio	1
4.4 Game Design	Panth Barola	1
2.1 Hover	Ndaru Bhramastra	2
2.4 Room Effects	Willem Ethan Tio	2
3.1 Text to Audio	Madhav Ajayamohan	3
3.2 Command to Audio	Madhav Ajayamohan	3
2.2 Background Color	Panth Barola	3
4.3 Background Music	Panth Barola	3

4.1 Hints	Willem Ethan Tio	4

Section 3.7 Sprint Two Code Review

No code review was completed, since no user stories were completed. Thus, no new branches need to be merged to the 'devBranch.'

Section 3.8 Sprint Two Retrospective

All members were in attendance for this sprint retrospective. This sprint was extremely unproductive, and none of the elements of the sprint goal could be achieved. One aspect that contributed to this unproductivity is the scheduling conflicts that each member had during the week. Every member had exams or projects from other classes due during that week, and therefore did not have enough time to complete their user stories. This reveals a lack of planning from our part for not taking into consideration our schedules.

We also decided to scrap User Story 1.5 because that can be implemented through MaryTTS which is another user story which helps accessibility.

Another aspect that led to our unproductivity is the difficulty in implementing the user stories. For example, user story 2.6 Trolls was far more difficult to implement than expected. Therefore, even though some work was done on the user story during the week, a usable troll game could not be produced by the end of the sprint.

Finally, there were also some issues with our group_37 repositories. As we had not created a .gitignore file, several unwanted files (out, .iml, etc.) were in the repository. The worst experience we had during this sprint was deleting these files from the repository. In order to not encounter this situation again, we decided to only commit files that we actually changed into the shared repository.

Section 4: Sprint Three Process Documentation

Section 4.1 Sprint Overview

The third sprint started on November 24, 2023 and ended on December 2, 2023. During this time period our goals are to:

- Implement the *addSliderEvent* method to change the brightness of the game screen,
- Implement the code to change the background color of each screen of the game,
- Implement the code to shift the screens of the game into a high contrast mode,
- Implement an event handler that checks when the mouse is hovering over an object and replaces the image of the object with a description of the object,
- Implement a replay audio button to replay a rooms description,
- Implement a health system for the Player,
- Implementing code to give the Player hints at certain rooms,
- Implementing code to process how a Player gets a buff or debuff via items or rooms,
- Implement the singleton design pattern to play music in the background of the game,
- Implement the abstract GameTroll class, and implement three concrete subclasses of GameTroll to create a troll game,
- Implement the code that rewards player achievements (e.g. visiting all rooms, fighting all trolls),
- Implement the method *readOutLoud* to read out every textual description,
- Implement the method *readOutCommand* to read out every command inputted by the user.

Section 4.2 Stories Selected for Sprint Three

User Story	Assigned to	Priority	Changes in Implementation
1.1 Brightness	Ndaru Bhramastra	1	The method implemented is *addSliderEvent* instead of *getBrightness*.
1.2 High Contrast	Panth Barola	1	All changes applied to updateScene().

1.3 Change Text	Madhav Ajayamohan	1	This user story was improved upon. Now, the text size of buttons will not change, and room descriptions will be displayed in a scrollpane so the entire description can be read, no matter the size.
1.4 Replay Sound	Ndaru Bhramastra	1	No changes
2.3 Health System	Willem Ethan Tio	1	Added an attribute to the abstract Player class that holds the lives of the current player.
3.3 Achievement List	Panth Barola	1	Changes made due to time constraint. Only one achievement implemented (Defeat Troll). This is implemented by checking if the special item is in inventory. This is checked by a boolean static
2.6 Trolls	Madhav Ajayamohan	1	The abstract class is now named *TrollState* and the implemented concrete class is *GameTrollState*. Furthermore, only one concrete subclass of *TrollState* will be implemented. The game will only have one type of troll, due to time constraints.
2.5 Item Effects	Willem Ethan Tio	1	Included a new method in AdventureGame that would switch the decorator wrapping the player.
2.1 Hover	Ndaru Bhramastra	2	No longer part of AdventureGameView, now part of TraversalState. Instead of changing the image, it now changes the item name with the description.
2.4 Room Effects	Willem Ethan Tio	2	Included a new method in AdventureGame that would switch the decorator wrapping the player.
3.1 Text to Audio	Madhav Ajayamohan	3	A method *say* was written in the AdventureGameView.java that will read a textual description. This method will be called in every instance text needs to be read.

			Due to time constraints, this user story was removed from the product backlog and not completed. Reasons explained in sprint retrospective.
3.2 Command to Audio	Madhav Ajayamohan	3	A method *say* was written in the AdventureGameView.java that will read a textual description. This method will be called in every instance text needs to be read. Due to time constraints, this user story was removed from the product backlog and not completed. Reasons explained in sprint retrospective.
2.2 Background Color	Panth Barola	3	The changes are directly made to the updateScene() method..
4.3 Background Music	Panth Barola	3	Changes reflected in updated UML diagram below
4.1 Hints	Willem Ethan Tio	4	No changes to implementation plan.

Section 4.3 Team Capacity

The goal is to complete all user stories by December 2, 2023. Thus, it will be possible to upload all the user stories on GitLab and merge all the code generated in sprint three by December 3, 2023.

Section 4.4 Collaborators

Collaborators	Responsibilities
Madhav Ajayamohan	User Story 1.3 Change Text User Story 2.6 Trolls User Story 3.1 Text to Audio User Story 3.2 Command to Audio
Panth Barola	User Story 4.3 Background Music

	User Story 2.2 Background Color User Story 1.2 High Contrast User Story 3.3 Achievement List:
Ndaru Bhramastra	User Story 1.1 Brightness User Story 1.4 Replay Sound User Story 2.1 Hover
Willem Ethan Tio	User Story 2.3 Health System User Story 2.4 Room Effects User Story 2.5 Item Effects User Story 4.1 Hints Merging ‘devBranch’ with the ‘main’ branch to signify all code for sprint three has been completed, and all components work together.

Section 4.5 Completed and Incomplete Tasks

The completed tasks include:

- User Story 1.3 Change Text
- User Story 2.6 Trolls
- User Story 4.3 Background Music
- User Story 2.2 Background Color
- User Story 1.2 High Contrast
- User Story 1.1 Brightness
- User Story 1.4 Replay Sound
- User Story 2.1 Hover
- User Story 1.3 Change Text
- User Story 4.2 Autosave
- User Story 2.3 Health System
- User Story 2.5 Item Effects
- User Story 4.1 Hints
- Merging ‘devBranch’ with the ‘main’ branch to signify all code for sprint three has been completed, and all components work together.

The incomplete tasks include:

- User Story 3.3 Achievement List

- User Story 2.4 Room Effects
- To create a new game file for our game.

Section 4.6 Product Backlog

User Story	Assigned to	Priority
1.2 High Contrast	Panth Barola	1
1.5 Audio Help	Panth Barola	1
3.3 Achievement List	Panth Barola	1
4.4 Game Design	Panth Barola	1
2.4 Room Effects	Willem Ethan Tio	2

Section 4.7 Sprint Three Code Review

Story Reviewed	**Name of Reviewer**	**Pull Request Link**
ID 1.1: Brightness	Willem Ethan Tio	https://mcsscm.utm.utoronto.ca/csc207_20239/group_37/-/merge_requests/8
ID 2.1: Hover	Willem Ethan Tio	https://mcsscm.utm.utoronto.ca/csc207_20239/group_37/-/merge_requests/11
ID 1.4: Replay Sound	Willem Ethan Tio	https://mcsscm.utm.utoronto.ca/csc207_20239/group_37/-/merge_requests/9
ID 2.3: Player Lives	Madhav Ajayamohan	https://mcsscm.utm.utoronto.ca/csc207_20239/group_37/-/merge_requests/7
ID 2.5: Special Power Ups	Madhav Ajayamohan	https://mcsscm.utm.utoronto.ca/csc207_20239/group_37/-/merge_requests/18

ID 4.1: Hints	Madhav Ajayamohan	https://mcsscm.utm.utoronto.ca/csc207_20239/group_37/-/merge_requests/13
ID 2.2: Background Color	Ndaru Bhramastra	https://mcsscm.utm.utoronto.ca/csc207_20239/group_37/-/merge_requests/17
ID 4.3: Background music	Ndaru Bhramastra	https://mcsscm.utm.utoronto.ca/csc207_20239/group_37/-/merge_requests/20
ID: 2.6 Trolls	Panth Barola	https://mcsscm.utm.utoronto.ca/csc207_20239/group_37/-/merge_requests/16

Section 4.8 Sprint Three Retrospective

In our third sprint retrospective, on December 2, all members were in attendance. This sprint was extremely productive, where a majority of all user stories in the sprint goal where eleven out of fifteen user stories were completed during the week.

Regarding user story 3.1 Text to Audio and user story 3.2 Command to Audio, the implementation of both were nearly completed. However, there were some issues with MaryTTS on Madhav's laptop, and thus the user story cannot be marked complete. In addition, the necessity of both user stories are being questioned. As *articulateRoomDescription* and *stopArticulation* work properly for all the new rooms, it may be unnecessary to do user story 3.1. Therefore, considering time constraints, there was not enough time to resolve the issues with user story 3.1 and user story 3.2, and both were removed from the product backlog.

User Story 2.4 was not fully implemented as it was still awaiting the creation of the game file that would contain the reformatted .txt files. As such, user story 2.4 was almost fully implemented and instead relegated to be completed in the final sprint alongside bug fixes.

All completed user stories were merged together in the 'devBranch,' and the code of all people were properly integrated to produce a working final product. There are some more components to add to the user story, however the project is mostly finished.

The one part of our process we should improve is coming to the meeting earlier– two of our members were half an hour to an hour late for the meeting. This will be something that we

will improve upon in our final sprint.

Section 5: Sprint Four Process Documentation

Section 5.1 Sprint Overview

The fourth sprint started on November 2, 2023 and ended on December 5, 2023. During this time period our goals are to:

- Implement the code to shift the screens of the game into a high contrast mode,
- Implementing code to process how a Player gets a debuff via rooms,
- Implement the code that rewards player achievements (e.g. visiting all rooms, fighting all trolls),
- Implement the *say* method *readOutLoud* to read out every textual description and every command inputted by the user,
- Address all potential bugs and fix any issues
- Implement Game Design by editing text files.
- Record a 10 minute video explaining each feature we implemented,
- Submit our final report and revised UML structures.

Section 5.2 Stories Selected for Sprint Four

User Story	Assigned to	Priority	Changes in Implementation
1.2 High Contrast	Panth Barola	1	HighContrastEffect is applied in the updateScene() methods of state directly.
3.3 Achievement List	Panth Barola	1	Changes made due to time constraint. Only one achievement was implemented (Defeat Troll). This is implemented by checking if the special item is in inventory. This is checked by a boolean static

			variable.
4.4 Room Design	Panth Barola	1	Changes made to text files. To create a new game file for our game. The new game files will include new features we implement, like a room with a troll into it, and rooms that give buffs or debuffs..
2.4 Room Effects	Willem Ethan Tio	2	No changes from previous implementation plan
4.3 Background Music	Panth Barola	2	Adds a ‘mute’ button to Settings to mute background music.

Section 5.3 Team Capacity

The goal is to complete all user stories by December 4, 2023. Thus, it will be possible to complete all tasks by December 5, 2023.

Section 5.4 Collaborators

Collaborators	Responsibilities
Madhav Ajayamohan	Review existing main code. Debug any errors. Continue unit testing. Record the 10 minute video. Edit and Format Report
Panth Barola	User Story 1.2 High Contrast - Bugs Fixed User Story 3.3 Achievement List User Story 4.4 Game Design Fix for User Story 4.3 BGM - Mute Button Review existing main code. Debug any errors. Continue unit testing.
Ndaru Bhramastra	Review existing main code. Debug any errors. Continue unit testing.

Willem Ethan Tio	User Story 2.4 Room Effects Merging ‘devBranch’ with the ‘main’ branch to signify all code for sprint three has been completed, and all components work together. Review existing main code. Debug any errors. Continue unit testing.

Section 5.5 Completed and Incomplete User Stories

The completed tasks include:

- User Story 2.4 Room Effects
- User Story 1.2 High Contrast
- User Story 3.3 Achievement List
- User Story 4.4 Game Design
- Fix for User Story 4.3 BGM - Mute Button
- Creating a new game file for the game (User Story 4.4)
- Record the 10 minute video.
- Finished Report

There are no incomplete tasks.

Section 5.6 Product Backlog

Nothing in product backlog, all user stories completed

Section 5.7 Sprint Four Code Review

Story Reviewed	**Name of Reviewer**	**Pull Request Link**
ID 1.2: High Contrast	Ndaru Bhramastra	https://mcsscm.utm.utoronto.ca/csc207_20239/group_37/-/merge_requests/23

ID 3.3: Achievements List	Ndaru Bhramastra	https://mcsscm.utm.utoronto.ca/csc207_20239/group_37/-/merge_requests/25
ID 4.4: Game Design	Ndaru Bhramastra	https://mcsscm.utm.utoronto.ca/csc207_20239/group_37/-/merge_requests/24
ID 2.4: Room Effects	Ndaru Bhramastra	https://mcsscm.utm.utoronto.ca/csc207_20239/group_37/-/merge_requests/22

Section 5.8 Sprint Four Retrospective

For our all last sprint retrospective, all members were present, and we celebrated the completion of our project. During the meeting, we spent time debugging any errors that occurred after combining all our user stories together, and eventually we obtained a working final product. To end our progress, we reflected on our work so far. We were proud of the final product we made, although there were some things we wanted to add that we couldn't due to time constraints. Our accomplishments and limitations are explained in further detail in the conclusion.

Overall, we were able to maintain our productivity from sprint three, and we held regular meetings over the sprint to debug, test and finalize the code. During this sprint, we managed to meet each other on time, unlike in previous sprints. The best experience in this sprint was finally being able to see our final game run properly.

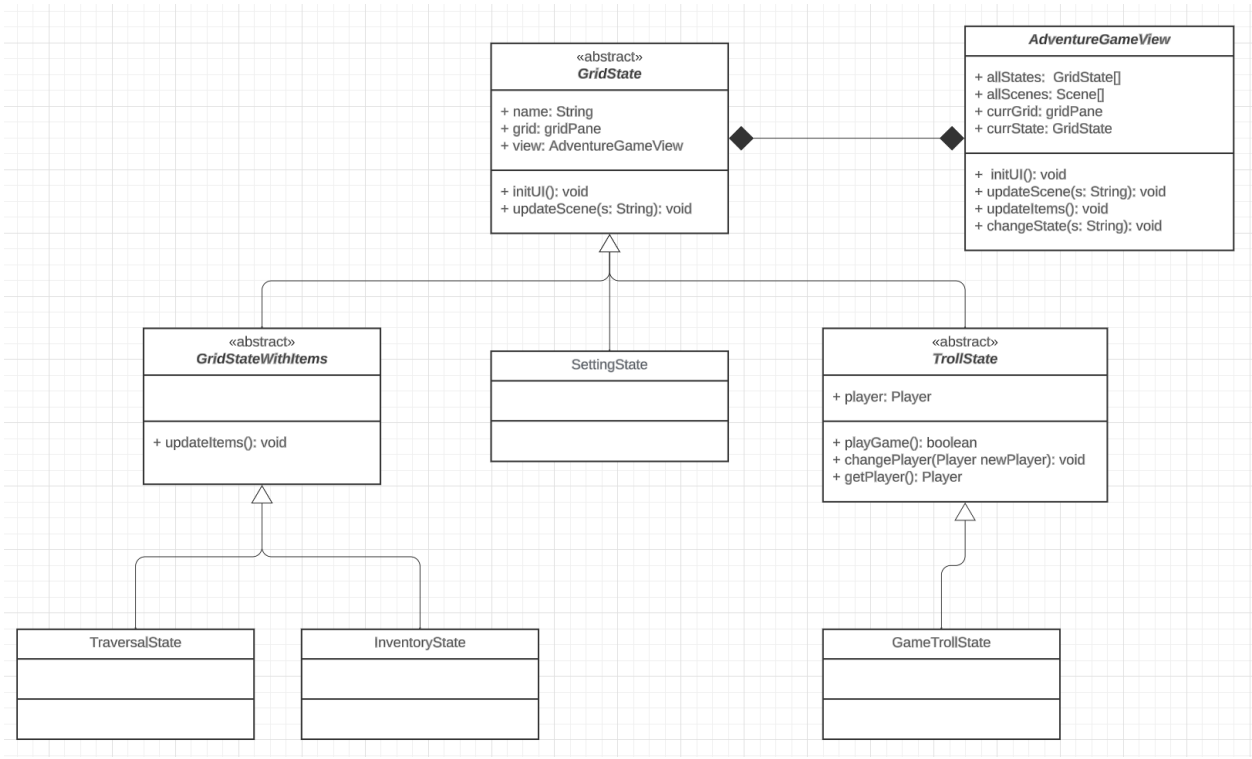
Section 6: Revised Design Patterns

Design Pattern #1: State Pattern

Implemented By: Madhav Ajayamohan

Overview: This pattern will be used to implement changing gridPanes.

UML Diagram:



Implementation Details: The UML Diagram outlines these main components:

1. The *GridState* abstract class, which includes
    a. Two attributes:
        i. name: the name of the grid pane
        ii. grid: the gridPane object
        iii. view: the AdventureGameView object the *GridState* object is contained in
    b. Four methods (more may be added):
        i. initUI(): Initializes the user interface of the gridPane
        ii. updateScene(s: String): Updates the gridPane after any changes
2. The *GridStateWithItems* abstract class, which is a subclass of *GridState*. This class has one method:
    a. updateItems(): Updates the list of items contained in the gridPane object of the *GridStateWithItems* objects
3. The *TrollState* abstract class, which includes
    a. An attribute, player: the attribute that represents the instance of the *Player* class that represents the user
    b. Three methods
        i. playGame(): this method returns true if the troll game is won, and false if the troll game is lost
        ii. changePlayer(Player newPlayer): this method changes the player attribute of a *TrollState* object to newPlayer
        iii. getPlayer(): this method returns the player attribute of the *TrollState* object

1. Some concrete subclasses of the *GridState* class
    a. *TraversalState*: represents the main screen where the user goes to different rooms
    b. *InventoryState*: represents the inventory of the player, that contains all object, power-ups and achievements of the user
    c. *SettingsState*: represents the settings of the game, where the user can change brightness, text size, contrast mode and background color of the game
    d. *GameTrollState*: represents a troll game that the user can play
2. The *AdventureGameView* class, which will now include:
    a. Four attributes:
        i. allStates: an array of all possible *GridState* objects for the specific instance of *AdventureGameView*
        ii. allScenes: an array of all possible *Scene* objects that correspond to the gridPane of each *GridState* object
        iii. currGrid: the current gridPane that is being represented on the screen
        iv. currState: the current *GridState* object whose gridPane is represented on the screen
    b. Methods:
        i. initUI(): Initializes the user interface of all *GridState* objects in allStates
        ii. updateScene(s: String): Updates the *currGrid* gridPane after any changes
        iii. updateItems(): Updates the list of items contained in the *currGrid* gridPane if *currState* is a *GridStateWithItems* object
        iv. changeState(String s): Changes the value of currGrid based on the parameter *s* that represents name of the *GridState* object

The design pattern will be used to switch the grid pane of the adventure game. Using the *changeState* method, the value of *currGrid* will be changed. Using the *isName* method of *GridState*, the *GridState* object that contains the new grid pane *currGrid* needs to be changed to be found. The new grid pane will be obtained using the *getGrid* method of *GridState*. The *initUI* method will be used to initially create the necessary grid pane.

If the gridPane of a grid state displays items– such as items in a room like *TraversalState* or items in the player inventory like *InventoryState*– this grid state will be defined as a *GridStateWithItems* object. The items on the *GridStateWithItems* will be updated with the *updateItems* method.

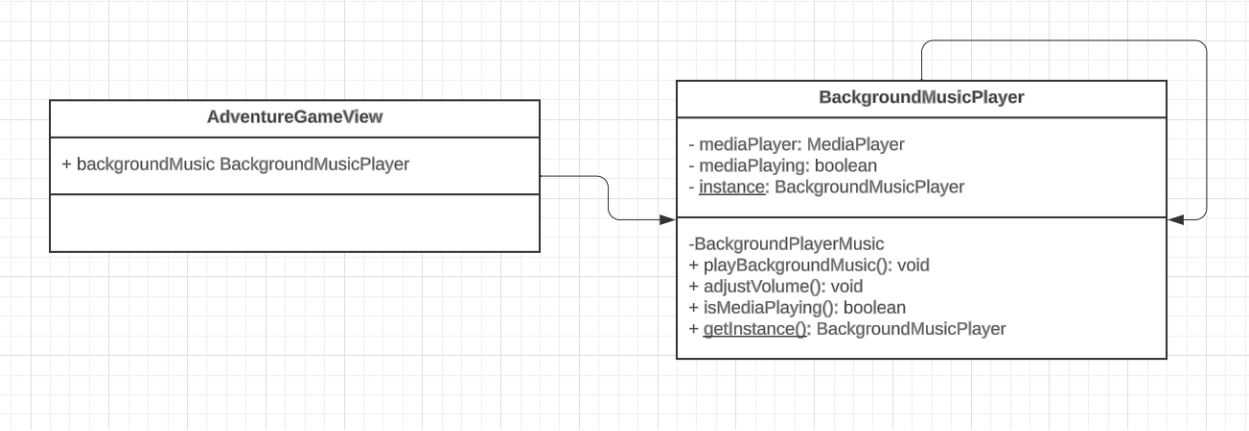
All concrete troll games will be implemented by creating a subclass of the *TrollState* abstract class. The game can be implemented in any way, and *playGame* will return true if the game is won and false if the game is lost. In order to facilitate the decorator pattern, the method *changePlayer* and *getPlayer* were added to *TrollState* in order to make sure the *player* attribute in the *TrollState* object represents the accurate buffed/debuffed player.

Design Pattern #2: Singleton Pattern

Implemented By: Panth Barola

Overview: This pattern will be used to implement background music.

UML Diagram:



Implementation Details: The UML Diagram outlines these main components:

1. 1. AdventureGameView
    a. Attribute
        i. backgroundMusic: object of BackgroundMusic
2. 2. BackgroundMusic
    a. a. Attributes
        i. mediaPlayer: object of MediaPlayer
        ii. mediaPlaying: boolean
        iii. Instance: BackgroundMusic object
    b. Methods
        i. playBackgroundMusic(): starts playing the background music which will be called as soon as the game starts.

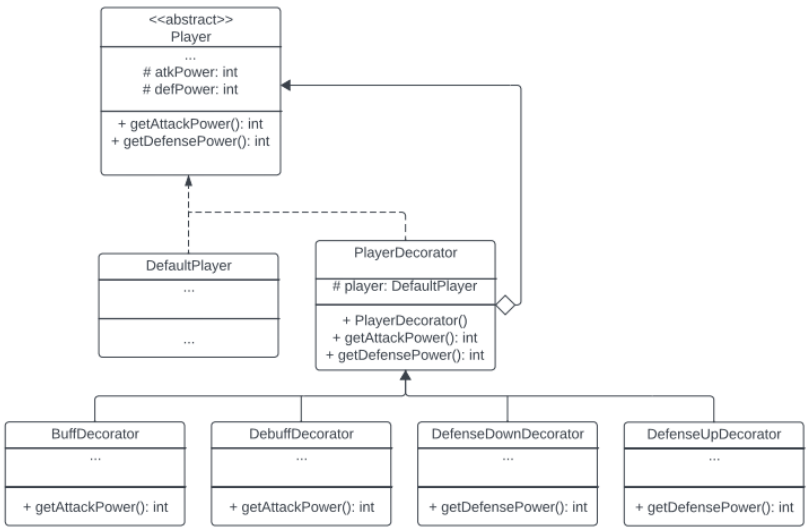
ii. adjustVolume(): changes the volume of the background music which is used to reduce volume when room description is playing and also when the background is muted.
iii. isMediaPlaying(): Returns if the media is playing to check if background music is playing and used to change the state from playing to pause and vice versa if required.
iv. getInstance(): Returns an object of BackgroundMusicPlayer to implement it using singleton.

This instance will be used throughout the running of the game with only this instance being affected by play and adjust the volume of music. It ensures that only one instance of BackgroundMusicPlayer is created and shared throughout the game. This instance is also a static variable. If an instance already exists, it returns that existing instance. This ensures that all parts of the game that need to control the background music are working with the same object. The audio is played at 0.8 volume (adjusted through the adjustVolume() method). However, when there is articulation of room description, the volume is adjusted to 0.1. Moreover, the mute function is implemented through a static variable which keeps a record of if it is muted or not. It checks if mute button is toggled and if so, it reduces the volume to 0.0.

Design Pattern #3: Decorator Pattern

Implemented by: Willem Tio

Overview: This pattern will be used to implement the autosave feature.



1. Player: abstract class that contains all the standard player attributes and methods
    a. Attributes
        i. atkPower: attack power of the player
        ii. defPower: defense power of the player
    b. Methods
        i. getAttackPower(): getter method for attack power
        ii. getDefensePower(): getter method for defense power

2. DefaultPlayer: Concrete default player object
    a. Attributes
        i. Inherits all the attributes from *Player*
    b. Methods
        i. Implements all methods from *Player*

3. PlayerDecorator: Concrete Decorator Class
    a. Attributes
        i. Inherits all attributes from *Player*
        ii. player: reference to concrete player object
    b. Methods
        i. Implements all methods from *Player*
        ii. PlayerDecorator(): Constructor
        iii. getAttackPower(): returns atkPower
        iv. getDefensePower(): returns defPower

4. BuffDecorator: Extends the base *PlayerDecorator*
    a. Attributes
        i. Inherits all attributes for *PlayerDecorator*
    b. Methods
        i. Inherits all methods from *PlayerDecorator*
        ii. Returns atkPower * 2

5. DebuffDecorator: Extends the base *PlayerDecorator*
    a. Attributes
        i. Inherits all attributes for *PlayerDecorator*
    b. Methods
        i. Inherits all methods from *PlayerDecorator*
        ii. Returns atkPower / 2

6. DefenseUpDecorator: Extends the base *PlayerDecorator*
    a. Attributes

        i. Inherits all attributes for *PlayerDecorator*
    b. Methods
        i. Inherits all methods from *PlayerDecorator*
        ii. Returns defPower * 2

7. DefenseDownDecorator: Extends the base *PlayerDecorator*
    a. Attributes
        i. Inherits all attributes for *PlayerDecorator*
    b. Methods
        i. Inherits all methods from *PlayerDecorator*
        ii. Return defPower / 2

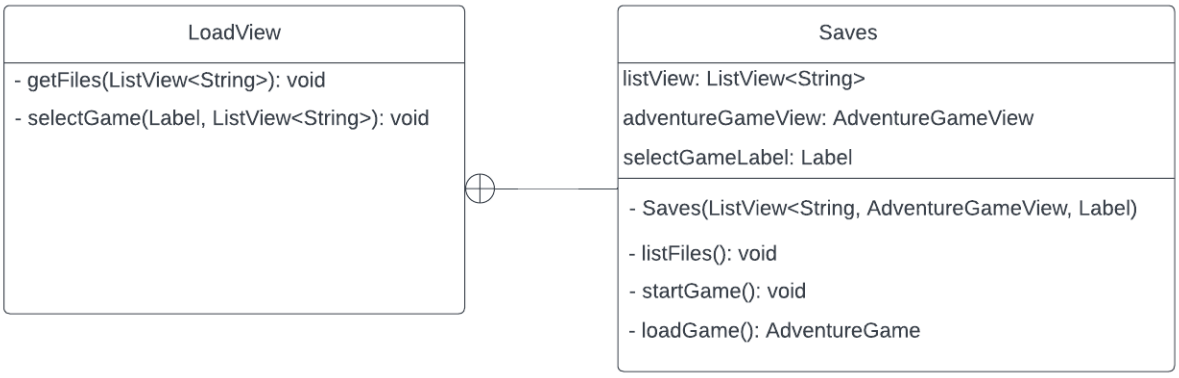
The *DefaultPlayer* class will act as the standard player object. The *PlayerDecorator* class is the base decorator that all specific decorators will inherit from. This class holds an attribute of the concrete player class. Each decorator extends the base decorator and implements their corresponding change. BuffDecorator double the atkPower, DebuffDecorator halves the atkPower, DefenseUpDecorator doubles defPower, and DefenseDownDecorator halves defPower. The new decorated player object becomes the new player that AdventureGame will refer to.

Design Pattern #4: Memento Pattern

Implemented by: Ndaru Bhramastra

Overview: This pattern will be used to implement the autosave feature.

UML Diagram:



Implementation Details:

1. LoadView

    a. Methods:
        i. The getFiles() method calls on the nested Saves class' listFiles() method.
        ii. The selectGame() method calls on the nested Saves class' startGame() method.
2. Saves
    a. Attributes:
        i. listView: the to-be populated list of save files (including autosaves and manual saves)
        ii. adventureGameView: the snapshot of the current AdventureGameView scene.
        iii. selectGameLabel: labels for successful and unsuccessful restoration of saves.
    b. Methods:
        i. Saves() constructor: initializes the Saves class to be used by the LoadView class.
        ii. listFiles() method populates the LoadView class' list of save files.
        iii. startGame(): restores the save file chosen and updates the current snapshot of AdventureGameView.
        iv. loadGame(): handles the actual loading of the chosen save file.

The LoadView class has a nested class called Saves that handles the restoring and populating of save files. This acts as the Memento class in the memento design pattern. Only the LoadView class can use the Saves methods, hence it only has private methods. The Caretaker class in the memento design pattern is not implemented as the folder system acts as a pseudo caretaker. The folder has all the save files and the Saves class accesses it to populate and restore the chosen save file.

Section 7: Conclusion

For everyone involved in this project, working on it was truly a wonderful experience. It gave us the opportunity to meet like-minded people, and the relationships we have built up through this project are truly invaluable. As we looked back on the project, there were several features that we were proud to complete, and several features we were not that proud of.

One feature that we are proud of was the implementing the State design pattern to implement the different screens of the game. While the knowledge required to implement the pattern was not complex, it required a good deal of effort and time in order to finally implement

this feature. However, the final product was quite exhilarating– particularly looking at how the screens can be seamlessly changed from one to another, and how each screen interacts with each other.

Another feature that we are proud of was implementing the Troll game. Initially, we were planning on making three unique troll games. However, due to time constraints, we could only develop one troll game. In addition, it was difficult to implement this singular troll game, particularly the part of the game that had to wait on user input. The implemented troll game is based off of a text based toll game developed for the first assignment. Therefore, it was hard to adapt the aspect of waiting for user input, and incorporating it into a while loop that loops until the troll's health points drop below zero or until the user's health points drop below zero. Eventually, this was resolved by throwing out the idea of a while loop! Instead, the implementation was as follows:

1. User enters a command in the TextField
2. Once the user hits enter, the method *submitEvent* is called to process the command
3. After the command is processed, the method *runTurn* is called to run a turn of the troll game
4. After *runTurn*, the method *playGame* method runs to check if, by the end of the run, the troll health points are zero or the player health points are zero. If either are true, the game appropriately ends.

Coming up with this solution, and seeing the troll game actually run was extremely satisfying as a developer.

On a more general note, due to the features being mostly GUI focused, it was a really enjoyable experience to learn about JavaFX and how much creative thinking the package allows. From simply changing brightness to setting multiple effects on the current grid and custom buttons, tinkering with JavaFX and getting more familiar with it is something we are all proud of.

Implementation of the power ups working in conjunction with other user stories was quite an interesting task. Since it required the culmination of multiple user stories together, it proved to be quite time consuming ensuring that all the different components of each method were working correctly. However, after numerous meets and hours of debugging, it was very satisfying to see all the pieces finally come together. The same idea also applies to the room

debuffs as both user stories work in a very similar fashion. The final result allows the player to be under the effect of one buff or debuff making the troll game much harder or significantly easier. Playing with these buffs makes the game more enjoyable and is something that ultimately became quite fun to program.

However, there were some things we could not implement, as we wanted. One example of this is how we could not create more troll games. Originally, we wanted three troll games, but with time constraints we could not add more troll games. Another feature that we wanted to implement better was regarding user story 3.1 (Text to Audio) and 3.2 (Command to Audio). By implementing these user stories, we would be able to improve the accessibility of the game. Every textual feature would be read out loud, and whenever the user touches a button, the game would notify the user that the user touched the button by sound. In addition, the game would read out any command the user inputted.

In fact, we were even able to see both of these user stories implemented. However, right before the implementation of both user stories were finished, an unidentifiable error appeared on the code. This error could not be fixed in time to integrate with all the other components of the game, and thus both user stories needed to be scrapped. This was a limitation of the project, and it is a feature we wished we could add within our project.

One of the limitations located within the player buffs and debuffs is the aspect that only one can be active at a time. As the power ups are implemented through a decorator design pattern, that amount of decorators and their roles were not fully finalized until much later in the work process. As a result, by the time we decided how these decorators would interact with AdventureGame, it was too late to add more decorators that would allow for multiple buffs and debuffs.

Another limitation is when implementing the Autosave and settings features, we were unable to save the settings the user has changed into the .ser file itself. This was partly due to a time constraint because of unforeseen difficulties and substantial code editing to the AdventureGame class.

Despite these limitations, we are still proud of the game we made, and we truly enjoyed the process. It has truly been an exhilarating experience.