Design Document

Version 1.1 – 2023.11.06

Created 2023.11.05

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

Project Identification

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.

User Stories

Name	ID	Description	Implementation Details	Priority	Effort
Ndaru	1.1	As a user with visual/auditory imparities, I want to be able to change the brightness of the screen to better interact with the game.	A slider will be present on the "Settings menu" and the pointer can be moved to the left to decrease brightness and to the right to increase brightness. The method *getBrightness* will convert the position of the pointer on the slider to a brightness value. An event handler will check for when the pointer is moved and will update the brightness accordingly.	1	2
Panth	1.2	As a user with visual/auditory imparities, I want to have access to a high contrast option to better interact with the game	highContrastMode will be a separate toggle button on the grid pane which when clicked would convert everything into high contrast mode. This will be implemented through changes in the *intiUI* method already implemented. A new private *toggleHighContrast* method will check if the highContrastMode toggle button is on or not and will assign the relevant style details in the *intiUI* method using an if-else statement and an *updateScene* is called after every toggle of the button to update the scene from one mode to another.	1	3
Maddy	1.3	As a user with visual/auditory imparities, I want to be able to change the size of the text to be able to clearly read the text.	On the 'Settings menu' of the game, there will be an ' Enlarge' and 'Minimize' button. These buttons will be handled by a *enlargeText* and *minimizeText* method. Upon a mouse click, the *enlargeText* method will increase the size of the text, and similarly *minimizeText* will decrease the size of the text. Additionally, *enlargeText* and *minimizeText* can also	1	2

			be called by clicking ‘Command/Control +’ and ‘Command/Control -’.		
Maddy	3.1	As a user with visual imparities, I want to be able to have audio descriptions for every textual description so I can understand the description	When the user presses ‘Command/Control P’ on the room traversal screen, the room description, and possible commands, will be read out loud. Similarly, when a user hovers over an object in the room traversal screen, the object description will be read out loud. When the user presses ‘Command/Control P’ on the help screen, the help text will be read out loud. In the inventory screen, when user hovers over an object or power-up, the description of the object or powerup is read out loud. These will be implemented by a method called *readOutLoud*. *readOutLoud* will be implemented using the MaryTTS Java library.	3	8
Ndaru	1.4	As a user with visual imparities, I want to be able to replay audio descriptions so that I can fully understand the description.	A”replay audio” button near the room description will be present. An event handler will check when the button is clicked. When clicked, the method *articulateRoomDesc* will stop all articulations currently running (if any) and play the room description audio.	1	2
Panth	1.5	As a user with visual imparities, I want to be able to hear sound effect when I choose a command so I can get audio confirmation that I chose the command.	This function will be implemented in *AdventureGameView* with public methods *soundForLook*, *soundForDrop*, and *soundForTake,* which will be called when the user inputs the commands look, drop and take, which are first handled by the *textEventHandler*. In case a invalid command, an invalidCommand sound will be played by calling the public method, *invalidCommandSound*.	1	3
Maddy	3.2	As a user with visual imparities, I want to be able to convert my text to speech, so that I can confirm if the command I entered is the command I wanted to enter.	When the user enters a command, the audio “Command [the command] was entered” will play. This will be implemented by the *readOutCommand* methods. *readOutCommand* will be implemented using the MaryTTS Java library.	3	8
Ndaru	2.1	As a user, I want to be able to see a textual description of an object when I hover over it, so that I can understand what the	Part of AdventureGameView Will have an event handler that checks when a pointer is hovering over an object. Replaces the image of the object with the description.	2	5

		object is.			
Panth	2.2	As a user, I want to be able to change the background color of the game, so that I can play the game in the color I want to.	The settings menu will have a select box with multiple options of colors and the background color changes according to the selection. Once the color is selected, the id of that color will be recorded to a global variable. The global variable will be used to override the style of the gridpane background color in the *intiUI* method from one color to another. *updateScene* will be called after every change.	3	2
Maddy	1.6	As a users, I want there to be different screens on the game such as: - A screen for traversing through rooms - A screen for the inventory - A screen for the help text - A screen for Troll games - A screen for the game settings so that I can be immersed in the game	This will be implemented using the State design pattern, as described below.	1	3
Maddy	1.7	As a user, I want to be able to have a 'Settings' menu, where I can change the visual/textual features of the game, so that I can interact with the game.	This will be implemented using the State design pattern, as described below. The 'Settings' menu will have buttons to change text, a slider to change brightness, a toggle switch for high contrast, and a select box to change background color.	1	3
Willem	2.3	As a user, I want there to be a three life system where the game is over after all lives are lost, so that the game is more interesting.	The Player class can gain a new attribute known as *HP*. A new method can also be created that either increases or decreases this value known as *lives*. Extra lives (from 2.5) can call this method.	1	2
Panth	3.3	As a user, I want to be able to be notified when I - Visit all locations in game - Defeat all Trolls - Defeat one of the trolls - Collect all objects so I can feel a sense of achievement.	A record is kept of all the rooms visited, number of objects in inventory and the count of trolls defeated with a notification popup once any of the achievements is completed and is recorded inside a menu as well. The records will be global integer variables named *roomsVisited*, *trollsDefeated* and *objectCounter*, keeping a count of the actions completed and comparing them to the total number of actions in the game. It will later utilize a *notify* method to create a pop-up which will inform the user if any of the achievements have been achieved.	1	5

Willem	2.4	As a user, I want there to be rooms with status effects that can damage me to make the game more interesting.	Specific rooms will contain methods that call upon the decorator design pattern to implement varying buff and debuffs. These methods will update the player's attack power and multiplier for a specified duration of time or until a specific condition is met.	2	3
Willem	2.5	As a user, I want to be able to gain power ups such as - Viewing what the next room is - Immunity to poison/paralysis/other status effect - Add 1 extra life - Increased attack power - Increased defensive power so that the game is more interesting	Similar to the method above, attack power and defensive power can be handled using the decorator design pattern. Running into objects that affect attack power can call upon the corresponding decorator class. Extra life can be handled by the *lives* method (in 2.3), and special items can have special classes that handle their functionality. A "*use*" object command can be created to call the object's special class which is then responsible for correctly handling the action of the object.	1	2
Maddy	2.6	As a user, I want to be able to face three different trolls, each with a different method to defeat, so that I can be more engaged.	As done in Assignment 1, the troll game will be implemented using the abstract GameTroll class. The three different trolls will be made using subclasses of the GameTroll class, each with different games. If the user loses against the troll, they will lose a life.	1	2
Wilem	4.1	As a user, I want to be able to get hints while playing the game so that it is easier to clear the game.	Rooms will have a new method *getHint* that will display a string giving the player a hint from this room.	4	1
Ndaru	4.2	As a user, I want to be able to autosave after I exit the game so that I do not lose all my progress.	Will be implemented using the Memento design pattern, as described below.	1	2
Panth	4.3	As a user, I want to be able to listen to background music during so that I can enjoy the game.	This will be implemented using the Singleton Design pattern where a single instance of class *BackgroundMusic* will be created when the game starts and plays out throughout the game.	3	2

Acceptance Criteria

Name	ID	Acceptance Criteria
Brightness	1.1	Moving slider pointer to the right increases the brightness Moving the slider pointer to the left decreases the brightness Brightness can only be changed using the pointer on the brightness slider. Brightness slider only appears on the "Settings menu"

Replay Sound	1.4	Plays the room description audio when “replay audio” button is pressed If button is not pressed, does not play the current room description audio Button is clickable Button is not clickable during forced room interactions
Hover	2.1	Shows the correct item’s description The text does not overflow outside of the object’s “pane” Object description replaces the object’s image only when hovered over the object Object image is returned when the pointer clicks on the object.
Autosave	4.2	AutoSave class saves the current state of the game when the game is closed *saveFile()* method does not override manually saved files. The user cannot override the autosave file.
Change Text	1.3	When a user clicks the ‘Enlarge’ button on the settings menu, all textual descriptions on different screens will be enlarged When a user pressed the ‘Command/Control +’ keys, all textual descriptions on different screens will be enlarged When a user clicks the ‘Minimize’ button on the settings menu, all textual descriptions on different screens will be minimized When a user pressed the ‘Command/Control -’ keys, all textual descriptions on different screens will be minimized
Different Screens	1.6	There will be a room traversal screen, where the player can traverse through different rooms The player can pick up objects in the room in the room traversal screen The player can move to the next room in the room traversal screen The player can save the game in the room traversal screen The player can load a new game on the room traversal screen There will be an inventory screen, which showcases all the player’s power ups and items The player can drop objects on the inventory screen by clicking on the object The player can hover over objects and power ups in the inventory screen and hear their text description There will be a help screen, that contains the instructions for the player, a legend for the possible commands of the user (like ‘Command +’ = enlarge text) There will be a settings screen, where the player can change the settings of the game, such as brightness, text size, the contrast of the grid pane, and the background color of the game There will be a screen for the troll games, where a game with the troll will be played It will be possible to switch between the room traversal screen, inventory screen, help screen and settings screen at will During a forced room interactions, none of the screens can be interacted with
Settings Menu	1.7	The menu has a slider to change brightness The slider can directly influence brightness of system The menu has an ‘Enlarge’ button that increases text size Clicking the ‘Enlarge’ button can increase the text size The menu has a ‘Minimize’ button that decreases text size Clicking the ‘Minimize’ button can decrease the text size The menu has a toggle switch of high contrast. Clicking the toggle switch will change to high contrast The menu has a select box that contains multiple background color options for the game

		Choosing a color in the select box can change the background color of the game
High Contrast	1.2	Change the theme of the game to high contrast as soon as the toggle switch is hit.
Audio Help	1.5	Every time a command is entered and accepted, it narrates the given command after every entry.
Trolls	2.6	When a room with a troll is entered, the screen will immediately switch to the troll game screen Each troll will have a unique game When a battle with a troll is lost, a player life is deducted When the player's life is lost, the game is forcibly ended
Text to Audio	3.1	On the room traversal screen, when the user presses 'Command/Control P', the room description and the commands possible in the room will be read out loud On the room traversal screen, when the user hovers over an object, the object description will be read out loud On the help screen, when the user presses 'Command/Control P', the help text will be read out loud On the inventory screen, when the user hovers over an object or powerup, the description of the object or power up is read out loud
Background Color	2.2	The background of the window changes to the selected color from the selection box.
Command to Audio	3.2	When the user enters a command in the room traversal screen, the audio "Command [the command] was entered" will play
Health System	2.3	When a player loses a TrollGame, their lives decrease by 1 When a player picks up an extra life, their lives increase by 1 When player lives reach 0, the player dies and the game ends
Room Effects	2.4	When a player enters a debuff room, their attack power is correctly reduced by a specific amount
Achievement List	3.3	The list in a textBox is updated every time a new achievement is achieved The style of the text changes from normal to having a strike through the middle.
Background Music	4.3	The Background music starts playing as soon as the game starts. The background music stops playing as soon as the game ends.
Item Effects	2.5	When a player picks up a buff/debuff item, their attack power is correctly increased /decreased by a specific amount When a player picks up an extra life, their lives attribute increases by one When a player picks up a special object, using the object correctly calls the special items methods
Hints	4.1	Clicking on the hints button or calling the hints command correctly displays the hint for the specific room

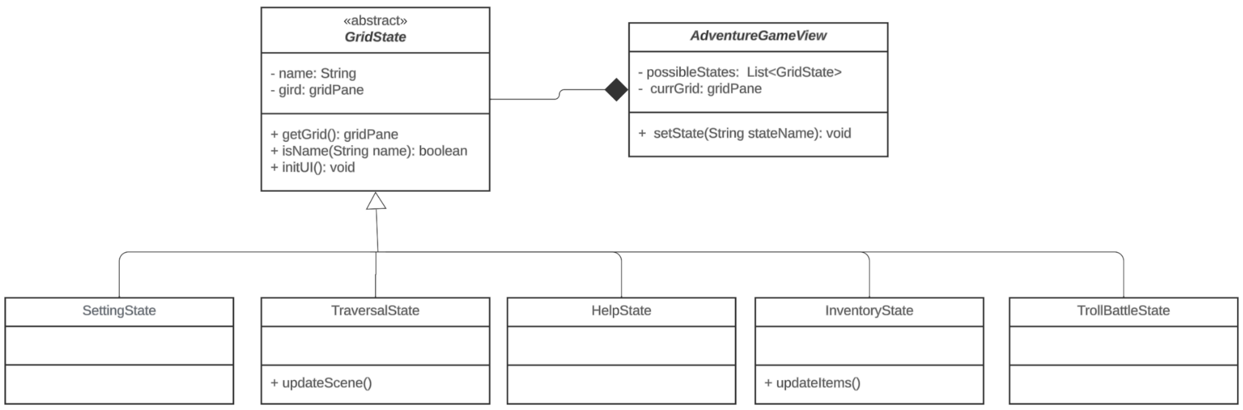
Design Patterns

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
    b. Four methods (more may be added):
        i. getGrid(): returns the gridPane object
        ii. isName(GridState g): Indicates whether the name of the GridState is "equal to" the name of this one.
        iii. initUI(): Initializes the user interface of the gridPane
2. Some suggested subclasses of the *GridState* class
3. The *AdventureGameView* class, which will now include:
    a. Two attributes:
        i. possibleStates: a list of all possible GridStates that the adventure game can have
        ii. currGrid: the current grid pane of the adventure game
    b. Methods:
        i. setState(String stateName): Changes the value of currGrid based on the parameter

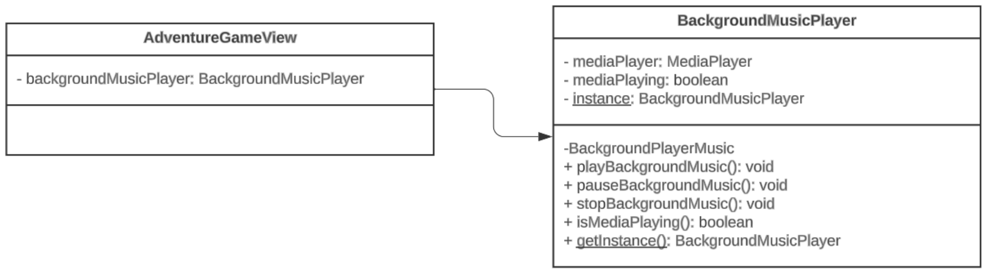
The design pattern will be used to switch the grid pane of the adventure game. Using the *setState* method, the value of *currGrid* will be changed. Using the *isName* method of *GridState*, the *GridState* object that contains the new grid pane *currGrid* needs to be changed to be found. The new grid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

Design Pattern #2: Singleton Pattern

Implemented by: Panth Barola

Overview: This pattern will be used to add a new functionality which is adding background music to the game.

UML Diagram:


AdventureGameView
- backgroundMusicPlayer: BackgroundMusicPlayer
BackgroundMusicPlayer
- mediaPlayer: MediaPlayer
- mediaPlaying: boolean
- instance: BackgroundMusicPlayer
-BackgroundPlayerMusic
+ playBackgroundMusic(): void
+ pauseBackgroundMusic(): void
+ stopBackgroundMusic(): void
+ isMediaPlaying(): boolean
+ getInstance(): BackgroundMusicPlayer

Implementation Details: The UML Diagram outlines these main components:

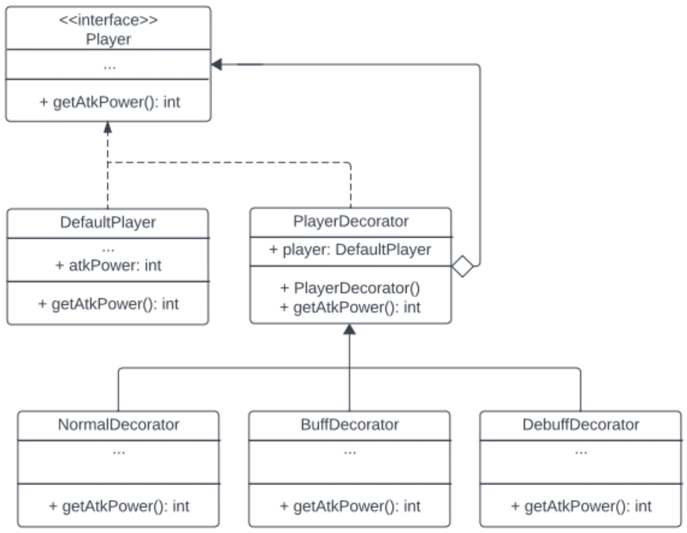
1. AdventureGameView
   a. Attribute
      i. backgroundMusicPlayer: object of BackgroundMusicPlayer
2. BackgroundMusicPlayer
   a. Attributes
      i. mediaPlayer: object of MediaPlayer
      ii. mediaPlaying: boolean
      iii. Instance: BackgroundMusicPlayer object
   b. Methods
      i. playBackgroundMusic(): starts playing the background music which will be called as soon as the game starts.
      ii. pauseBackgroundMusic(): pauses the background music which will be called if any narration or in-game activities require muting of the music.
      iii. stopBackgroundMusic: stops the background music will be called during the end of the game or when quitting before the autosave function.
      iv. 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.
      v. 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 pause, play and stop music. It ensures that only one instance of *BackgroundMusicPlayer* is created and shared throughout the game. 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 *BackgroundMusicPlayer* class is implemented as a singleton to ensure there is only one instance responsible for managing the background music throughout the game. The Singleton Pattern here guarantees that there is only one instance of the *BackgroundMusicPlayer*, and all parts of the game can easily control and interact with the background music without worrying about multiple instances causing synchronization issues or inconsistencies.

Design Pattern #3: Decorator Pattern

Implemented by: Willem Tio



Implementation Details:

1. Player: contains all the standard player attributes and methods
    a. Has one additional method getAtkPower()

2. DefaultPlayer: Concrete default player object
    a. Attributes
        i. Inherits all the attributes from *Player*
        ii. atkPower: int (default starting attack power)
    b. Methods
        i. Implements all methods from *Player*
        ii. getAtkPower(): returns the correct calculation for atkPower

3. PlayerDecorator: Concrete Decorator Class
    a. Attributes
        i. Inherits all attributes from *Player*

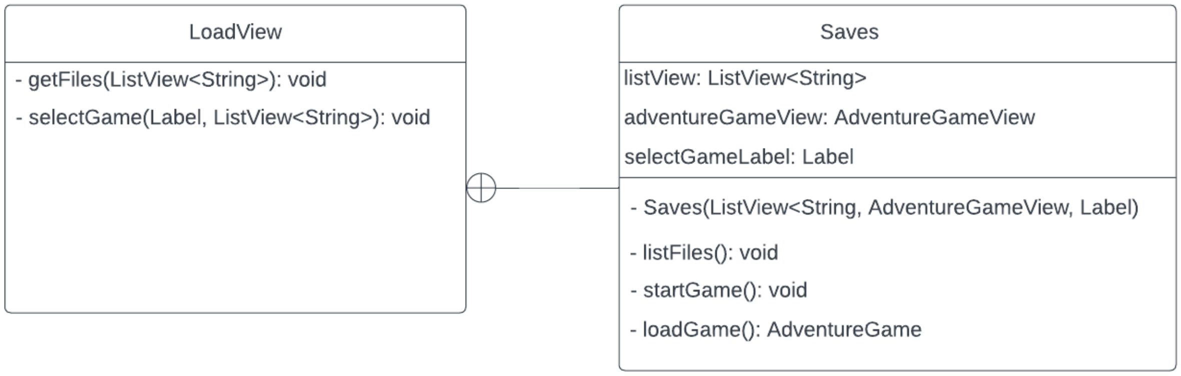
    ii. player: DefaultPlayer
  b. Methods
    i. Implements all methods from *Player*
    ii. PlayerDecorator(): Constructor
    iii. getAtkPower(): returns the correct calculation for atkPower

4. Normal…/Buff…/DebuffDecorator: Extends the base *PlayerDecorator*
    a. Attributes
        i. Inherits all attributes for *PlayerDecorator*
    b. Methods
        i. Inherits all methods from *PlayerDecorator*
        ii. Each decorator contains their own specific implementation of getAtkPower(): return the correct calculation for atkPower

The *DefaultPlayer* class will be the player object that is altered by the decorator pattern. The *PlayerDecorator* class acts as the base class for all decorators which holds an attribute that references the player object to be changed, which in this case is the *DefaultPlayer*. Each of the respective decorators that extend the *PlayerDecorator* class will have their own implementation of *getAtkPower()*. This method will return the calculated attack power for the player based on their current buffs and debuffs. When the *DefaultPlayer* needs to be updated, *DefaultPlayer* can change to reference a new decorator object corresponding to the correct status effect. Thus, when *getAtkPower()* is called, it will return the correct value.

Design Pattern #4: Memento Pattern

Implemented by: Ndaru Bhramastra


LoadView
- getFiles(ListView<String>): void
- selectGame(Label, ListView<String>): void
Saves
listView: ListView<String>
adventureGameView: AdventureGameView
selectGameLabel: Label
- Saves(ListView<String, AdventureGameView, Label)
- listFiles(): void
- startGame(): void
- loadGame(): AdventureGame

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.