Design Document

Version 1.0 – 2023.11.06

Created 2023.10.31.

Project Name

Name 1 Adriana Gaydov
Name 2 Abdullah Enaya
Name 3 Jason Chen
Name 4 Jose Pestana

GitLab Repository:

https://mcsscm.utm.utoronto.ca/csc207_20239/group_74

SECTION 1: PROJECT IDENTIFICATION

The primary motivation for undertaking this project is to gain experience and knowledge in project management, teamwork and game development. The project encourages creativity and collaboration, dedication and perseverance. It encourages applying technological and theoretical knowledge in a practical setting.

The enhancement this project will bring to the existing functionality are aimed at making the game more accessible and engaging. By implementing features such as minigames, boss fights, character classes, sound effects and background music, the main goal is to improve the players' gaming experience. These features will not only improve the overall quality of the user experience, but potentially increase player satisfaction.

Accessibility is a crucial part of the project. Accessibility is incorporated in the project in several different ways: color themes, text-to-speech and speech-to-text.
The project is made to be not only fun and engaging, but also inclusive and accessible to all players.

In essence, the aim of the project is to create an accessible and engaging game as well as to encourage personal growth.

SECTION 2: USER STORIES

Our **user stories** are given below:

Name	ID	Owner	Description	Acceptance Criteria	Implementation Details	Priority	Effort
Save and Load Game	1.1	Adriana	As a player who has in-game progress, I want to be able to save the game under a unique name, so that I could easily identify and load my saved game later, enabling me to continue from the point I left it off.	The player is able to save their game progress under a unique name. The player is notified when they try to save a game with a name that already exists. After successfully saving a game, a confirmation message is displayed in the graphical interface. The player is able to load a game and continue it from the point they left it off. All saved games are displayed in a list of saved games, identified by their name, date of creation and last date of modification.	Create Save and Load views that display all the saved games in a list format (ListView). For each saved game, store the name, character class, date of creation and last date of modification as an item of the ListView. For each view create a button for save, load and implement an event handler. A game is saved by serializing an object and a game is loaded by deserializing an object. Create and update a label for displaying success/fail messages.	1	1
Delete Game	1.2	Adriana	As a player who has saved games, I want to be able to pick a saved game from a list of all my saved games and have the option to delete it, so that I can remove saved games I do not want to play anymore.	The player is able to select a saved game from a list of all saved games. Once a game is selected, the player is able to indicate via the graphical interface that they want to delete the game. Before deleting a saved game, the player is asked for confirmation to avoid accidental deletions. The confirmation message should clearly indicate the name of the saved game to be deleted.	Create a Delete view that displays all the saved games in a list format (ListView). In the view, create a button for delete and implement an event handler. Create a Confirm view that confirms whether the player wants to delete the game or not.	1	1

				Upon deletion, the game file is removed from the list of saved files and from the storage. A message is displayed indicating that a game has been successfully deleted.	The Confirm view displays a message in a label and has two buttons: Yes/No. The Yes/No buttons have an event handler. If Yes button is pressed, delete the file using File.delete() and display an appropriate message in a label in the Delete view.		
Load Game Model	1.3	Adriana	As a player who enjoys different game stories, I want to be able to pick a game model from a list of all available games so that I can play the game I want.	The player is able to select a game model from a list of all available game models. After the player has selected a model, the game model is loaded, and a new game is started. A message is displayed indicating that a game model has been successfully loaded and a new game has started.	Create a SelectModel view that displays all the available game models in a list format (ListView). For each model, store the name of the model as an item of the ListView. Create a button for load model and implement an event handler. Display a success message (in a label) and start a new game.	1	1
Speech-to-text (Accessibility)	1.4	Adriana	As a player who is unable to use a keyboard or a mouse, I want the game to have a robust voice command feature (speech to text) and to execute all the commands I have said out loud, so that I am able to enjoy the game without the need for traditional input devices.	The player is able to turn on the speech to text functionality by saying a keyword. The game should inform the player about the possible commands via the graphic interface. The player is able to say out loud a command and play the game without using a keyboard or mouse. The game displays the command and prompts the player to confirm the command. The game should record the player's command, when confirmed, and execute that command as intended. The player is aware that a command has been recorded and executed via the graphic interface.	Use Google's Speech-to-Text API. The functionality is turned on when the player says, "Activate speech to text". Create a ConfirmCommand view that displays in a label the requested command and has 2 buttons: Yes/No for confirming the command. Add event handlers to the Yes/No buttons. If yes is pressed, execute the command and notify the player via the graphical interface (e.g., with label text). Otherwise, wait for the player's new command.	2	3
Colour Templates	1.5	Adriana	As a player with specific colour preferences, I want the ability to customise the game's colour scheme by selecting from a variety of pre-designed colour templates, so that I can personalise my gaming experience and ensure that my colour preferences align with the graphic interface of the game.	The game provides a list of all available colour templates. The player is able to select a colour template from the list. The option to change the colour template is accessible from the game interface (e.g., via Settings). The game remembers the player's selected colour template. Accessibility: Some of the available colour templates are distinguishable by colorblind players.	Create a variety of pre-designed colour templates that cater to different colour preferences. Ensure some templates are distinguishable by colorblind players. Create a Settings view that enables to player to select a colour theme from a list of all available colour themes. Once a player selects a colour template, apply that template to the game's graphic interface. This could involve changing the colours of various game elements such as backgrounds, text, buttons, etc. Store the player's selected colour template (e.g., a Settings class will keep the current colour template).	1	3

Sound effects	1.6	Adriana	As a player who enjoys immersive gaming experiences and sound effects, I want the game to play unique and engaging sound effects when I perform key actions such as levelling up, adding an item to my inventory, or engaging in a boss fight, so my gaming experience is enhanced by providing auditory feedback that corresponds to my actions and progress in the game.	The game has unique sound effects for each key action such as levelling up, adding an item to an inventory, engaging in a boss fight. The sound effect corresponding to a key action is played, serving as auditory feedback. The sound effects have high quality, they are clear and distinct, so that confusion and distraction is avoided. The player is able to mute/unmute the sound effects (e.g., via Settings). Accessibility: The game should be fully playable and enjoyable when the sound effects are muted, to cater to players with hearing impairments or those who prefer to play without a sound.	Create unique and engaging sound effects for each key action in the game such as levelling up, adding an item to an inventory, or engaging in a boss fight. The sounds come with each game model in “effects”/” sounds” directory. Provide an option in the game settings to allow the player to mute or unmute the sound effects according to their preference. Play the sounds when the player levels up, fights with a boss, picks up/drops an item, enters a room, etc.	1	3
Lives/HP	1.7	Adriana	As a player who thrives on challenges, I want the game to incorporate a lives and health points (HP) system (such that if my HP drops to 0, I lose a life and if I lose all my lives, I lose the game), so that I am encouraged to play more strategically, make careful decisions to preserve my HP and lives, and prolong my survival in the game.	The game displays the player’s current health points (HP) in the graphical interface. The game displays the number of lives the player has remaining in the graphical interface. During a boss fight, if the player’s HP drops to 0, the game triggers the loss of a life and provides appropriate feedback (sound effect, message). If the player has 0 lives remaining, the player loses the game and is prompted to either restart or exit the game.	Update the game’s graphical interface to display the player’s current HP and the number of lives remaining. This could be in the form of a health bar and a lives counter (as part of the game interface). HP is displayed only during boss fights. Lives are displayed throughout the whole game. HP is reset back to full after the player loses or wins a boss fight. If a player’s HP drops to 0 during a boss fight, reduce the number of lives by 1 and restart the boss fight. If the player loses all lives, show a message (in a label) informing that they lose. Prompt the user to select Restart/Exit game. The lives system can be implemented with a separate class. The hp is an attribute of character.	1	2
Text-to-speech (Accessibility)	1.8	Adriana	As a player that is visually impaired, I want the game to incorporate text-to-speech functionality in various English dialects, including but not limited to American and British English, so that I am able to fully engage with the game, independent of its graphical interface.	The game has a text-to-speech feature that can be easily enabled/disabled by the player (e.g., via Settings). The text-to-speech feature supports multiple English dialects. The player is able to select their preferred English dialect from a list of all available options. The player is able to navigate (using tab key) and interact with the game entirely through audio cues, without the need to see the graphical interface.	Use MaryTTS to implement the text-to-speech functionality. The functionality is enabled in the Settings menu. All the available dialects are shown in a list and the player is able to select one. An event listener and handler will apply the selected dialect and enable the text-to-speech. All graphical components must have alt text and should be all linked (as	3	3

				All graphical objects have alternative text to support the text-to-speech feature.	nodes). Request focus is used to enable “tabbing” between different elements.		
Challenges	1.9	Abdullah	As a player, I want to encounter challenging puzzles, minigames, or boss fights that lead me to treasures, important items, or certain rooms, and gaining me XP, so that I get a sense of challenge in the game, making it more fun.	To access certain areas, or to pick up certain items, the player is made aware that there is a challenge that needs to be completed before they are able to access them. The player can choose to start the challenge, where it will then begin. The player could not engage with the challenge. Winning the challenge awards the player with what they were trying to access. The player is also rewarded XP. Some challenges are puzzles or minigames, where the player can play the game and either win it in time or lose and have to go back where they started from. Some challenges are boss fights, where the player can lose HP, or lives if they lose the fight, where they return where they started. Once a certain area or item’s challenge has been beaten, the player has free access to it, as it has been unlocked. Some rooms, when entered, start an unprompted boss fight, as the player would not have knowledge of the boss before entering.	Different states of the AdventureGame model are implemented using the State pattern. Paths can be blocked by a Minigame or Boss, similar to Trolls, where the paths in the file would indicate a Minigame or Boss blocking the path. It is then stored as the keyName in the Passage. When viewing possible paths, the blockage is also shown, and it could be described in the room description. If the command received is to a Passage that has a blockage, it would instantiate a GameState of that kind, and switch the state of the game to it. The GameState would take in the Minigame or Boss, then instantiate the view for it, and store it in the state, where it can be grabbed by the AdventureGameView by calling getMainView. It would then set it to the middle of the view. The Minigame and BossFight classes handle the mechanics of the challenge, where they would take in any action from their Views and handle them in the specific game. A Minigame, for example, would handle the inputs by moving the sprite of the player or its environment accordingly, and calculate a win or a loss. If a win occurs, the Passage is then unlocked, and the player is moved to the according new Room, or the item is added to the player inventory. If it is an item, then that AdventureObject would have an attribute storing what game needs to happen before it is picked up.	1	3
Boss fights	1.10	Abdullah	As a player engaged in a boss fight, I want to have a challenging fight with a boss, where I can use my abilities and items to win the fight, and where losing would lose me a life, so that I am incentivized to fight strategically.	During the boss fight, the player engages in turn-based attacks against the boss. HP of the boss and the player are both in view. The player can utilize their weapons or any items they have during the fight. The player can utilize abilities during the fight, only ones that are specific to fighting. The player is shown possible moves during their turn, including items or abilities to use.	A BossFight class handles this. A BossFightView class updates the view as the fight is happening. The BossFight takes in a Boss and stores it as an attribute, where a Boss has HP, attack damage, and any other abilities it might have as attributes. BossFight handles turns, giving control to the player in one turn, then letting the Boss attack in the next. When a player attacks, it calls the attack method from the player’s Character. That method looks at equipped items and applies buffs accordingly. When a defense is called, the damage taken by the player decreases for the next time.	2	3

Levelling up	1.11	Abdullah	As a progressing player, I want to level up throughout the game through gaining XP, where levelling up would allow me to progress to new parts of the game and give me more abilities, eventually allowing me to win the game, so that I find progressing rewarding.	The player can see their current level and XP points at all times. After reaching a certain amount of XP points, the player's level increases and a new XP counter appears, representing the amount of XP between the current level and the higher level. Levelling up could grant the player new abilities or better stats. The player is notified about the level change, and any new abilities they gain. The player is made aware of rooms or items inaccessible to them at the current level, and the level they would be accessible at. Certain items give the player XP when first acquired. Certain rooms give the player XP when first discovered. Winning challenges awards the player with XP. A certain room is only accessible at a certain level, that would allow the player to win and finish the game.	Player stores a level attribute. Level would be a class that stores the xp and current level, and handles switching levels when the right xp is reached. It stores info about levels. The game View creates Nodes that represent the xp and level. All passages have a levelNeeded attribute, which store which level is needed before the passage is accessible. When accessing possible actions, the game shows what level is needed for the path if it is not yet reached. As the player wins minigames or boss fights or picks up certain items xp is added to the player using an addXP method, which does the needed checking for leveling up. Increments the level attribute when needed. The View then updates and displays a sound effect and message when the level is updates. AdventureObjects have an xp attribute storing how much xp will be given to the player on first pickup.	1	2
Map	1.12	Abdullah	As a player, I want to have a map I can view that tells me where I am as well as the rooms I have discovered, so that I can traverse the game more easily.	The map is dark in the beginning, only showing the current room. As the player enters new rooms, the map expands with rooms the player enters, as well as pathways accessible from those rooms.	A MapView is implemented, which has a Map. As the player moves to new Rooms, which store their coordinates on the map, the Map illuminates that part. There would be a picture of the full map, and then it would be overlayed with black, except the parts that are discovered.	3	
Character Classes	1.13	Jose	As a player starting the game, I want the ability to choose character classes with unique skill sets so I can play to my liking	Upon first launch, the game must prompt the player to choose between a set of classes. Each class will have universal features (basic movements and attacks) Each class type must have their own set of skills. Each class strength / weakness and play style will be made apparent to the player. Class types fight and interact with mobs differently. Classes will each be given a unique look. Each class should effectively progress through the game in a similar matter (will all have the access to the same challenges and rooms)	Implemented by the factory design pattern. A base character interface will be made with universal moves to be shared by all unique character classes. A character factory class will be created to return the base character interface. Character classes such as mage, fighter and other class types will implement the base character interface, as well as their own unique set of skills.	1	3

Character abilities	1.14	Jose	As a player who values change and improvement, I want to gain new abilities for my character as I play so that I can feel a sense of progression.	Ties into the level and XP of the player's character System must have an ability menu. Shows all the abilities currently available to the player as well as unlockable ones the player will receive as they progress in levels (i.e., x amount of experience pints to unlock / will unlock at x level) The menu will show an in-depth description of each ability. Ability icons will also be shown on the main game HUD. The abilities can be used with the 1,2,3,4… keys. Players will be able to see what abilities are available to them when fighting mobs. Abilities are unique to the player class and are unable to use another classes'	Uses the XP/Level feature of the game. Each class will be given a predetermined set of abilities which are unlocked at specific levels. The levels of each character will be tracked throughout the game, so that when needed level is achieved, the player will be given the option to unlock that ability.	1	2
Armour and Weapons	1.15	Jose	As a player who values customization and progression, I want the ability to use different weapons/armour so that I can accompany my class style of play.	System must be able to validate an iheatem is usable by a specific class (i.e., class specific items) Must implement a menu for players to see current equipped items. Will have the ability to interchange between a variety of armour and weapon equipment. Armour will consist of a helmet, main garment, leggings, and boots, with an accessory (ring, necklace etc.) the main weapon will vary depending on class. Armour and weapons should boost player stats (health, attack damage)	Uses the XP feature of the game. Items such as weapons or armour will be dropped by mobs. Items will be made character specific, and usable to the player at a specific level. Armour and weapons will be given a stat attribute which will boost player stats (health, attack...). An inventory feature will be added to characters to store items.	2	2
Background Music	1.16	Jason	As a player who enjoys immersive video game experiences, I would like the game to include a variety of background music in order to build a common theme across an adventure and for certain rooms to have different moods.	The game should prompt users to enable or disable background music. Volume should be adjustable. Desired Soundtracks should play when entering a room or encountering a challenge. There should be no abrupt changes in background music when soundtracks change. Background Music should not overpower (play louder than) sound effects and narration.	Dependent on the Settings Class, the game will determine whether to play or not to play music and at what loudness to play music at. Room and Challenge Specific Music Files will be stored in game files and identified by file name. If there are no specified music files, default music may be play instead. Using the Media and MediaPlayer Class of the JavaFX library, the game will load, play and transition between sound files as needed.	2	1

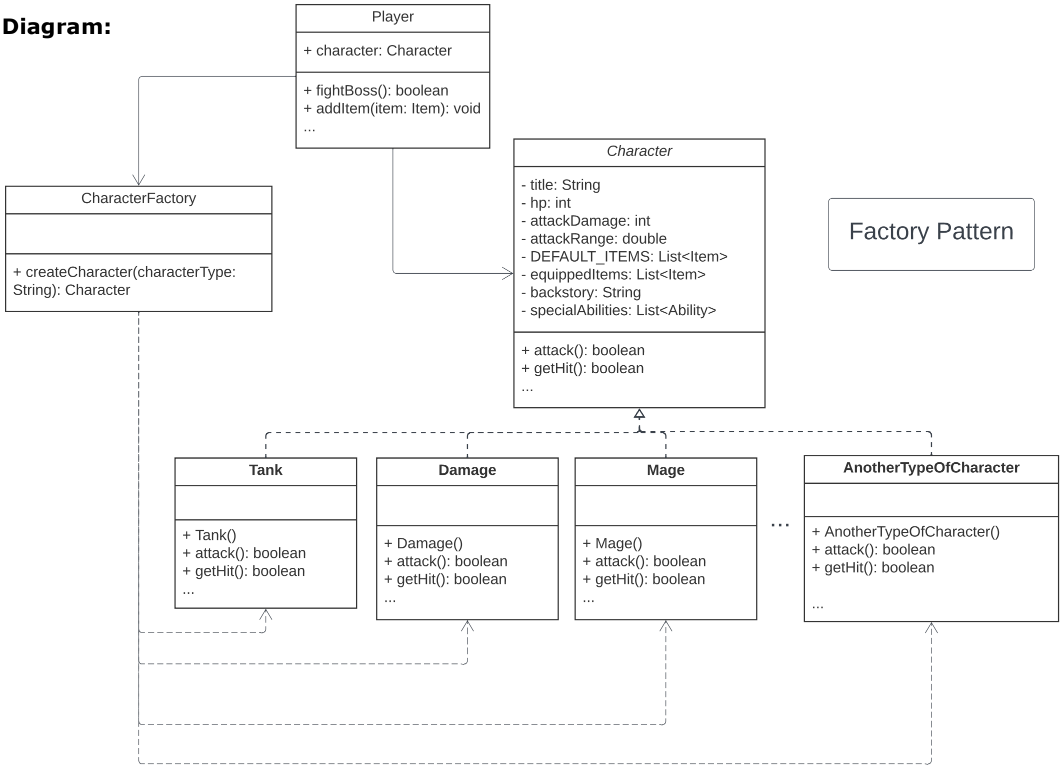
Passive Objects and Characters	1.17	Jason	As a player who enjoys many interactions, I would like the game to include a variety of non-violent characters and objects such as animals and levers that can be interacted but whose effect may or may not be a central part of the game in order to add opportunities for interaction which helps create detailed and consistent in-game world.	The option to interact with the object should be accepted when an object is in the room. The object could return a message. The object could return an item. The object could engage in dialogue. The object could modify the in-game world in some way. Adventure designers should be able to specify how objects are presented in the game files (The object is hidden, item is needed to interact, etc.)	Implemented by the Decorator Design pattern. Many different concreteComponents such as characters, animals and objects along with a baseDecorator implement a single entity, which utilizes one or more of the baseDecorators possible subclasses to add distinct behaviour to the entity. For more details see Decorator Pattern UML Diagram Below.	1	3
Developer Mode	1.18	Jason	As an adventure designer, I would like a developer mode where I can skip, select and play certain rooms, engage in certain challenges and interact with objects without restrictions based on health points, experience points or in-game obstacles such as certain items being required or locked behind world mechanisms.	Adventure designers should be able to enter developer mode. Adventure designers should be able to select and fast travel to rooms. Adventure designers should be able to select and fast travel to challenges. Adventure designers should be able to input any valid command bypassing all restrictions. If there are multiple results to valid commands, the game should warn and proceed to the most advanced scenario. Adventure designers should be able to play any challenge ignoring all restrictions. Adventure designers should be able to interact with any objects ignoring all restrictions. Adventure designers should be able to set their own player statistics such as health points and experience points. Adventure designers should be able to obtain any valid items.	Dependent on settings, developer mode is either enabled or disabled. In developer mode, users may input any valid movement command regardless of restrictions. The DeveloperMode class interacts with the AdventureGame in order to force a move. In developer mode, users may input special commands to modify their health, experience points and inventory, the DeveloperMode class handles all these requests and passes them to the required classes. In developer mode, user may interact with any objects including chests and characters, the DeveloperMode class handles all these requests and passes them to the Chest class and the InteractableElement class. In developer mode, users may replay or defeat challenges by using special commands, the DeveloperMode class handles all these requests and passes them to the MiniGame class. In developer mode, users may use special commands to travel directly to locations, the DeveloperMode class handles all these requests modifies the Player Class to move the players.	3	2

SECTION 3: SOFTWARE DESIGN

Design Pattern #1: Factory Pattern (Adriana)

Overview: This pattern will be used to implement different kinds of characters with individual and personalised gameplay during boss fights.

UML Diagram:



Implementation Details: The UML diagram outlines these main components:

- The *Character* abstract class, which includes 8 common attributes: title, hp, attackDamage, attackRange, DEFAULT_ITEMS (constant), equippedItems, backstory and specialAbilities, and 2 methods: attack and getHit.
- The concrete classes Tank, Damage, Mage, …, AnotherTypeOfCharacter (if needed). Each class will have personalized title, hp, attackDamage, attackRange, DEFAULT_ITEMS (constant), equippedItems, backstory and specialAbilities. Each class will also have different implementation for the methods attack and getHit.
- The class CharacterFactory will produce a specific type of character (Tank, Damage, Mage, …) depending on the player’s request.
- The class Player has a character attribute and is associated with the CharacterFactory that will produce the requested type of character.

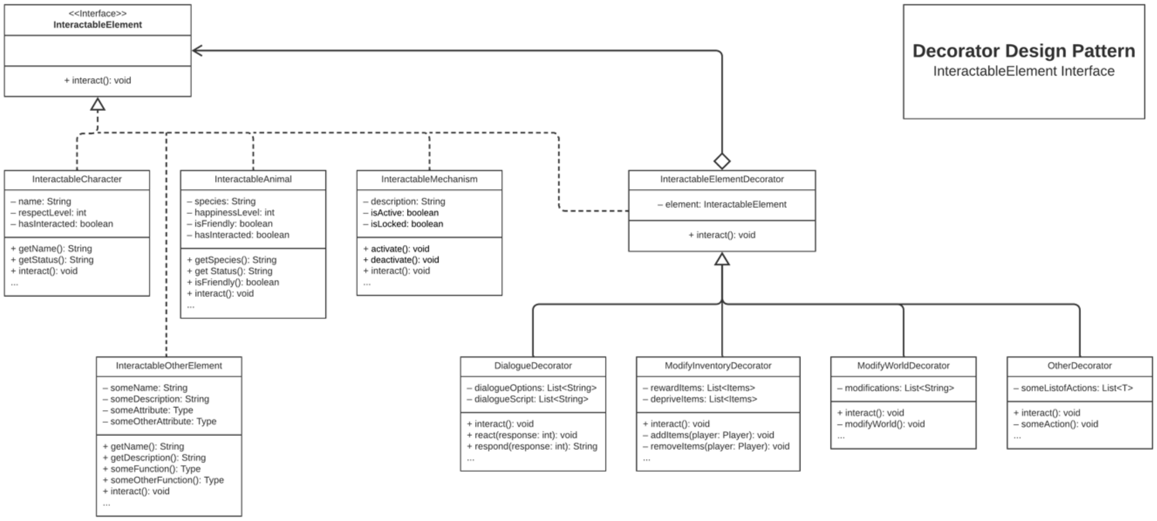
The CharacterFactory requires a string indicating the character type (title) and creates and object of the selected type of character. In the beginning of the game, the player is prompted to select a character via the graphical interface (e.g., from a ListView). Then, the factory creates an object of that type of character using the string received from a graphical component (e.g., a ListView). This object is set as the character attribute in the Player class.

Each subclass of class Character (Damage, Tank, Mage, …) will override the parent’s constructor by setting different, personalised for the character type, title, hp, attackDamage, attackRange, DEFAULT_ITEMS (constant), equippedItems, backstory and specialAbilities. For instance, the Tank class will have more HP and less attack damage, whereas the Damage class will have less HP and more attack damage. Each subclass will also implement differently the attack and getHit methods. For instance, the Tank class's getHit method will reduce the HP less than the Damage class's getHit method.

Design Pattern #2: Decorator Pattern (Jason)

Overview: This pattern will be used to implement various passive characters and objects who may possess one or more distinct behaviours.

UML Diagram:



Implementation Details:

The UML Diagram above followed the basic structure of a decorator design pattern. The main components of the UML Diagram are ConcreteComponents, BaseDecorator and Concrete Decorators. ConcreteDecorators are classes which defined basic attributes and behaviours that can be modified by the decorator classes, in this application, these are animals, characters, objects or some other entity.

The BaseDecorator serves a foundation for all ConcreteDecorators (in this application, they are connected via an inheritance relationship) and serves to facilitate modifications to behaviours of ConcreteComponents from ConcreteDecorators, in this application, the BaseDecorator is called the interactableElementDecorator.

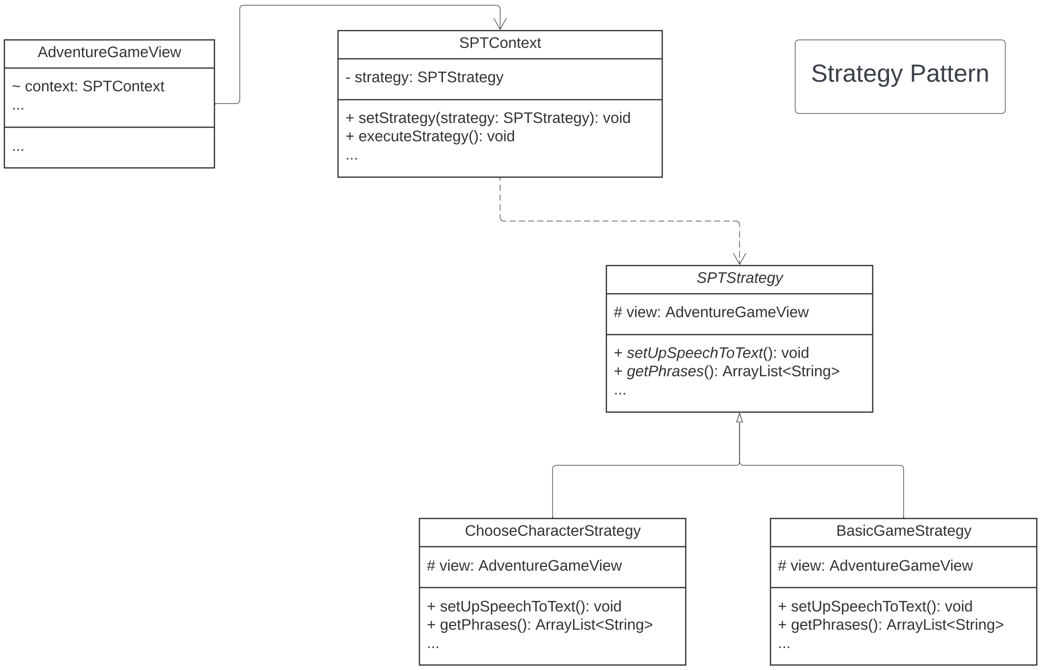
ConcreteDecorators are classes that modify and enhance the behaviours of BaseDecorators, in this application they primarily modify the interact method in additional to adding functionality such as more complicated dialogue and in-game world modifications.

This application of the decorator pattern allows for different ConcreteComponents to be created and its behaviour determined through the possible combinations of subclasses of InteractableElementDecorator each adding its own distinct behaviour to the object, allowing different entities to *mix and match* behaviours.

Design Patter #3: Strategy Pattern (Jose, Adriana)

Overview: This pattern will be used to implement the Speech-To-Text feature of the game.

UML Diagram:


AdventureGameView
~ context: SPTContext
...
...
SPTContext
- strategy: SPTStrategy
+ setStrategy(strategy: SPTStrategy): void
+ executeStrategy(): void
...
Strategy Pattern
SPTStrategy
# view: AdventureGameView
+ setUpSpeechToText(): void
+ getPhrases(): ArrayList<String>
...
ChooseCharacterStrategy
# view: AdventureGameView
+ setUpSpeechToText(): void
+ getPhrases(): ArrayList<String>
...
BasicGameStrategy
# view: AdventureGameView
+ setUpSpeechToText(): void
+ getPhrases(): ArrayList<String>
...

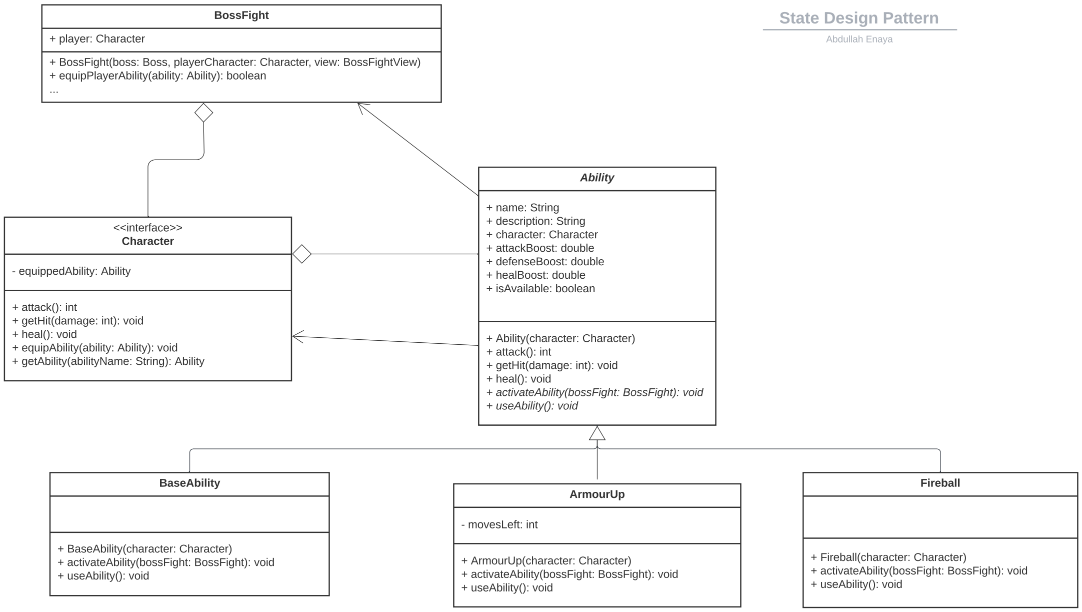
Implementation Details: The UML diagram outlines these main components:

- The SPTContext class that contains the strategy attribute and the methods setStrategy and executeStrategy
- The abstract class SPTStrategy that contains the attribute view and the methods setUpSpeechToText, getPhrases.
- The subclasses ChooseCharacterStrategy and BasicGameStrategy that implement the specific strategy for setting up the speech-to-text

The game view contains a speech-to-text context that is updated using the method setStrategy when the game transitions from the character selection to the basic game. Depending on the context, either ChooseCharacterStrategy or BasicGameStrategy is executed. The accepted phrases for each strategy, as well as the setup of speech-to-text, are specific to the strategy.

Design pattern #4: State Design Pattern (Abdullah)

Overview: This pattern will be used to implement the different states of the ability of the character during a boss fight.


State Design Pattern
Abdullah Enaya
BossFight
+ player: Character
+ BossFight(boss: Boss, playerCharacter: Character, view: BossFightView)
+ equipPlayerAbility(ability: Ability): boolean
...
<<interface>>
Character
- equippedAbility: Ability
+ attack(): int
+ getHit(damage: int): void
+ heal(): void
+ equipAbility(ability: Ability): void
+ getAbility(abilityName: String): Ability
Ability
+ name: String
+ description: String
+ character: Character
+ attackBoost: double
+ defenseBoost: double
+ healBoost: double
+ isAvailable: boolean
+ Ability(character: Character)
+ attack(): int
+ getHit(damage: int): void
+ heal(): void
+ activateAbility(bossFight: BossFight): void
+ useAbility(): void
BaseAbility
+ BaseAbility(character: Character)
+ activateAbility(bossFight: BossFight): void
+ useAbility(): void
ArmourUp
- movesLeft: int
+ ArmourUp(character: Character)
+ activateAbility(bossFight: BossFight): void
+ useAbility(): void
Fireball
+ Fireball(character: Character)
+ activateAbility(bossFight: BossFight): void
+ useAbility(): void

Implementation details: The UML diagram outlines these main components:

The BossFight class, which includes the method equipPlayerAbility and the attribute player.
The Character class contains an equippedAbility attribute and has an equipAbility method that modifies the current ability (current state).
The abstract class Ability contains the methods attack, getHit, heal, activateAbility, and useAbility. The subclasses of Ability are the three types of ability during a boss fight: BaseAbility, ArmourUp, Fireball.

The player is able to equip a certain ability that can be used when the player requests. Each type of ability has a different implementation, and the Character class maintains the current equipped ability. The ability is activated with the abstract method activateAbility in class Ability.