FINAL Project Report

Created 2023.11.17

<u>Touch Grass</u>

Moeez Omair

Adam Ebid

Nikhil Iyer

Youssef Samaan

<u>Table of Contents:</u>

SECTION 1: REPORT SUMMARY

The scope of this project was to create an Adventure Speedrun game inspired by the concept of experiencing nature in a virtual environment. It is a running joke amongst computer science students that we are always glued to our computer screens and never get outside and “Touch Grass”. As such, we decided to create a stimulating and fun video game to do just that.

The project aims to build upon knowledge from this course’s knowledge, assignments as well as previous computer science courses to develop a game of our own. We have built upon a foundation from our assignment 2 implementation in order to make the base of this game.

Our intention was to create a game that's not just enjoyable to play, but also a testament to the range of skills, knowledge and learning opportunities we've gained from working as a group.

In this report, we embrace the Agile process by showcasing the highlights of each sprint iteration, including completed user stories from the product backlog, participation of each team member. We also included a section for updated versions of our UML design patterns from the Phase 1 report, providing the structure we used for certain aspects of the game.

In the final section, we briefly summarize both our project accomplishments and its limitations as well as our final thoughts and takeaways from this collaboration experience and environment.

SECTION 2: PROCESS DOCUMENTATION

SPRINT 1 OVERVIEW

2.1.1 Sprint Overview:

Preceding our first sprint, the group set up a Discord server where we can maintain constant communication and collaborate where we encountered issues with coding tasks, we were responsible for. Once this was set up, we decided to embark upon our first sprint.

For this sprint, our goal was to have a working grid where we can build all our other features off. This provides us with a foundation where we can simulate game from the player's perspective and have a visual of how the features and accessibility features add onto the game in real time.

2.1.2 Stories Selected for this Sprint:

- Game Grid
- Move Player
- Save Game
- Load Game
- Grass/Poison Ivy Generation
- Player HP Bar
- Gaining Money
- On-Screen Money Display

2.1.3 Team Capacity:

We expect to complete these selected stories by October 17th, so there is no delay in the implementation of the rest of the game.

2.1.4 Participants:

Our sprints were all conducted individually, while we collaborated together on Discord to help one another with issues we faced and coming up with suggestions. The highest levels of productivity we had seen was during group meetings in voice calls as we were able to communicate with one another in real-time.

Moeez Omair:

- Organizing provided and additional resources
- Maintaining a medium of communication (Discord)
- Writing documentation and reports for the sprints
- Save/Load Game

Adam Ebid:

- Project Master (merging all the changes in the end)
- Game Grid
- Player Movement

Nikhil Iyer:

- Grass/Poison Ivy Generation
- Multiple Grass Sprites

Youssef Samaan

- Player HP Bar
- Gaining Money
- On-Screen Money Display

2.1.5 Tasks Completed:

What was actually finished during this sprint?
During this sprint, we have structured a majority of our base game as per the specifications presented in

2.2. SPRINT 1 PRODUCT BACKLOG

- Save/Load Game

- Text To Speech
- Item Shop

2.3. SPRINT 1 CODE REVIEWS

Story Reviewed	**Name of Reviewer**	**Pull Request Link**
Grid-Making	Adam	Here
FoliageGen	Nikhil	Here
Save/Load Game	Moeez	Here
PlayerHP	Youssef	Here
Gaining Money On-Screen Money Display	Youssef	Here

2.4 SPRINT 1 RETROSPECTIVE

When a sprint is completed, hold a retrospective meeting and ask one team member to take notes. Place a short record of each retrospective meeting in this section! The details should include:

- The participants in the meeting;
    - All group members: Moeez, Nikhil, Adam, Youssef
- Any unfinished tasks;
    - Save/Load Game
    - Text To Speech
    - Item Shop
- A summary of practices that went well this sprint and should be continued;
    - Discord server was the best idea:
    - Used the text chat and the voice chat to the fullest
    - Pushing onto different branches so we can all work together
- A summary of new or revised practices to include moving forward;
    - Making sure we communicate when we run into problems
- A summary of any bad practices that will not be repeated moving forward;
    - N/A
- Your team's best/worst experience during this sprint
    - JavaFX kept deleting itself when we switched branches so we needed to manually fix it every time

SPRINT 2 OVERVIEW

2.1.1 Sprint Overview:

During our second sprint, our goal was to flesh out all our work with features, some of which were for accessibility. There were a few user stories which carried over to this sprint which we needed to work on. Overall, we were succcessful in doing so and had a much more accurate sense of time taken this time around.

2.1.2 Stories Selected for this Sprint:

- **Save Game (continued)**
- **Load Game (continued)**
- **Restart Game**
- **Colourblind (accessibility)**
- **Sensory Overload**
- **Inventory**
- **Item Shop**
- **On-Screen Timer**
- **Death Screen**

2.1.3 Team Capacity:

The team aims to complete the selected stories by the set deadline of October 24th, ensuring a smooth transition into subsequent phases of the project.

2.1.4 Participants:

Once again, our sprints were conducted individually and we had moments of collaboration in between.

Moeez Omair:

- Writing documentation and reports for the sprints
- Save/Load Game continued
- Restart Game

Adam Ebid:

- Project Master (merging all the changes in the end)
- Overall UI
- Red/Green Colorblind Filter
- Sensory Overload

Nikhil Iyer:

- Cutting Grass
- Player Take Damage

Youssef Samaan

- Death Screen
- Inventory
- Item shop

2.1.5 Tasks Completed:

We finished all the the user stories presented above with room for improvement and further modifications based on later code changes

2.2. SPRINT 2 PRODUCT BACKLOG

N/A - We've completed the deliverables for this sprint.

2.3. SPRINT 2 CODE REVIEWS

We're expecting that each team member will make some changes to the team repository at each sprint (meaning we expect to see roughly weekly commits). Moreover, we're expecting that before changes on feature branches are transferred to your team's develop branch, that your team will conduct code reviews. You can do these in class! Each team member should provide at least one code review for one of their peers at each sprint iteration. Your reviews will be documented in your repository, but we ask that your briefly document them here as well using this format:

Story Reviewed	**Name of Reviewer**	**Pull Request Link**
Red/Green Colorblind Filter	Adam	Here
PlayerTakeDamage	Nikhil	Here
Cutting Grass	Nikhil	Here
Restart Game	Moeez	Here
Save/Load Game (complete)	Moeez	Here
Death Screen	Youssef	Here
On-Screen Time	Youssef	Here
Inventory Item Shop	Youssef	Here

2.4 SPRINT 2 RETROSPECTIVE

When a sprint is completed, hold a retrospective meeting and ask one team member to take notes. Place a short record of each retrospective meeting in this section! The details should include:

- The participants in the meeting;
    - All group members: Moeez, Nikhil, Adam, Youssef
- Any unfinished tasks;
    - N/A
- A summary of practices that went well this sprint and should be continued;
    - Collaborating in times of difficulty
    - Pushing our code only in working condition
    - Making sure we didn't overburden ourselves
- A summary of new or revised practices to include moving forward;
    - Be sure you are balancing CSC207 project with other university responsibilities
- A summary of any bad practices that will not be repeated moving forward;
    - N/A
- Your team's best/worst experience during this sprint
    - Merge branch ended up breaking the code due to a merge conflict and we needed to manually fix it

SPRINT 3 OVERVIEW

2.1.1 Sprint Overview:

Finally, we wanted to make a few more additions but it was close to time for wrapping up. There were things in our backlog which we didn’t get to but as our project stood by the end of this sprint, we had a complete game. This sprint was a hassle when it came to merging all our branches but eventually, we figured it out.

2.1.2 Stories Selected for this Sprint:

- **GameAudio**
- **End of Game Behaviour**
- **Updated Red/Green Filter (accessibility)**
- **Enhanced Grid Borders (accessibility)**
- **Optimizing Assets**
- **Polishing UI**

2.1.3 Team Capacity:

We expect to complete these selected stories by December 2nd, so there is no delay in the implementation of the rest of the game.

2.1.4 Participants:

Moeez Omair:

- Optimizing/Compressing Assets
- Leading group organization and distributing finishing tasks
- Writing documentation and reports for the sprints
- (Updated) Save/Load Game
- (Updated) Red/Green Filter
- Enhanced Borders Filter
- Optimize Assets

Adam Ebid:

- Project Master (before Nikhil took over)
- Better UI buttons

Nikhil Iyer:

- Project Master (merging all the changes in the end)
- Game Audio
- JavaDoc Comments

Youssef Samaan

- Improved UI Layout
- Updated Money Class

2.1.5 Tasks Completed:

What was actually finished during this sprint?

During this sprint, we have structured most of our base game as per the specifications presented in

2.2. SPRINT 3 PRODUCT BACKLOG

There were a couple of user stories which we didn't come around to implement at the end:

- Text to Speech
- Treasure Progress Bar
- Tool Upgrade System (this became inventory and item shop)
- Save and Load Game (we removed this as it led away from the intended focus of the game)
- Improved Audio (for proximity to poison ivy)

2.3. SPRINT 3 CODE REVIEWS

Story Reviewed	**Name of Reviewer**	**Pull Request Link**
Updated Red/Green Filter	Moeez	Here
EnhancedBorders	Moeez	Here
Optimized Assets	Moeez	Here
Improved UI Layout	Youssef	Here
Updated Money Class	Youssef	Here
Better UI Buttons	Adam	Here

2.4 SPRINT 3 RETROSPECTIVE

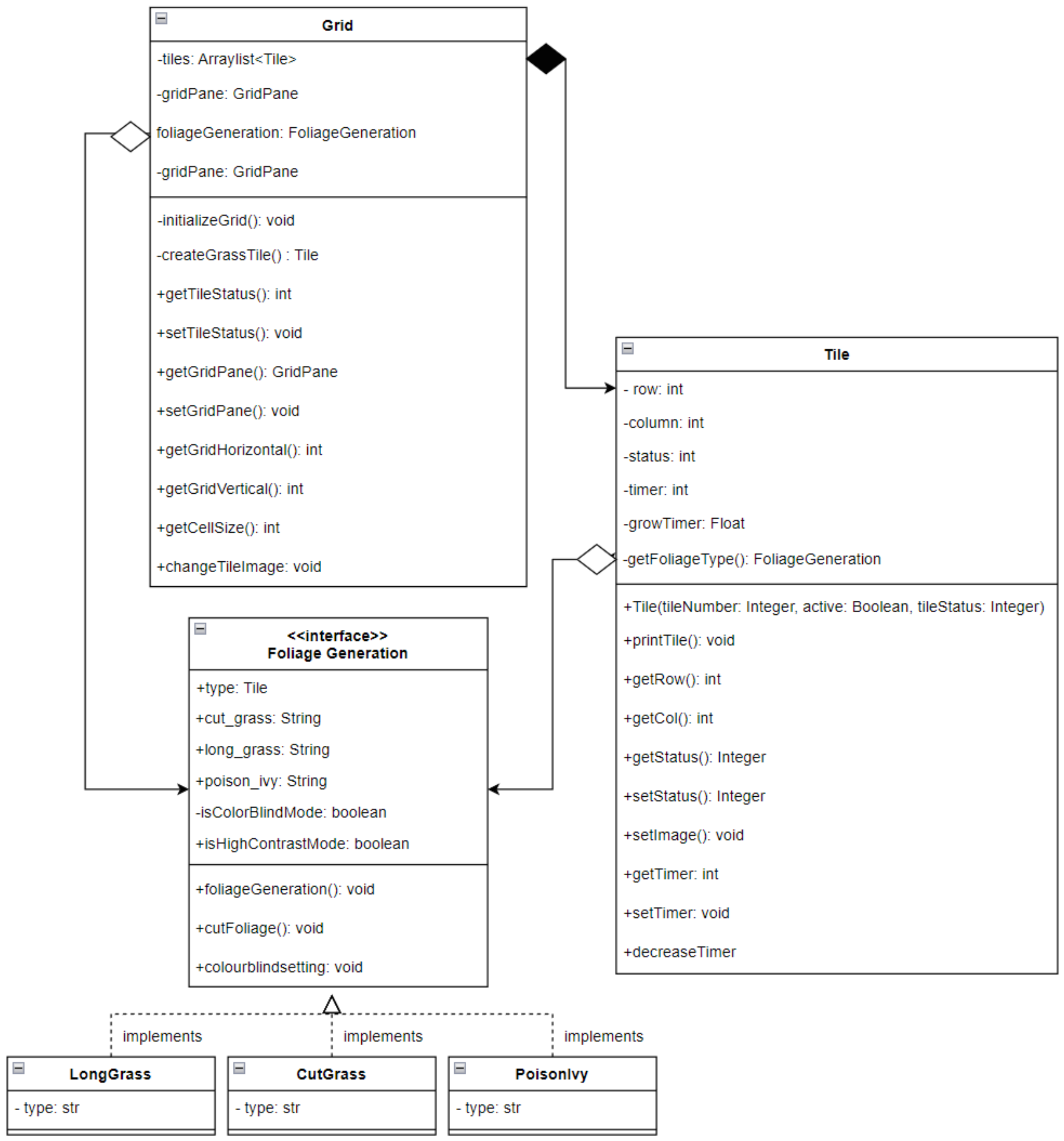
- The participants in the meeting;
    - All members were present: Moeez, Nikhil, Adam, and Youssef
- Any unfinished tasks;
    - There were some tasks we wish we would have been able to complete but perhaps if we expanded this game later on
- Your team’s best/worst experience during this sprint
    - Merging went a lot worse this time around, we had to manually merge multiple branches line by line and that took us hours together in a voice call

SECTION 3: UPDATED UML DESIGN PATTERN

Design Pattern #1: Builder and Decorator

UML

Moeez – Builder and Decorator



Implementation Details

Revising my original UML, I have realized that a "builder" and "decorator" patterns reflect my design choices best after implementing everything. The UML diagram above explains the implementation of the grid's appearance.

Grid:

The Grid class is responsible for grid where all Tile objects are presented, collected within an ArrayList. There will be an active tile attribute and a method to extract it.

Tile:

The Tile class represents each individual tile within the grid. The tile class has multiple attributes related to its implementation as well as methods to extract it. There are also additional methods required internally to keep the method functional. FoliageGeneration is also used to further make this function complete.

Foliage Generation:

The FoliageGeneration interface provides a contract for different types of foliage that can decorate a tile, such as long grass, cut grass, and poison ivy. Moreover, when implementing the accessibility features, this allows us to apply the filters to each of the tile's photos with ease.

LongGrass, CutGrass, PoisonIvy:

These classes implement the Foliage interface, each representing a different type of foliage that can exist on a tile. Each of these have unique properties, such that:

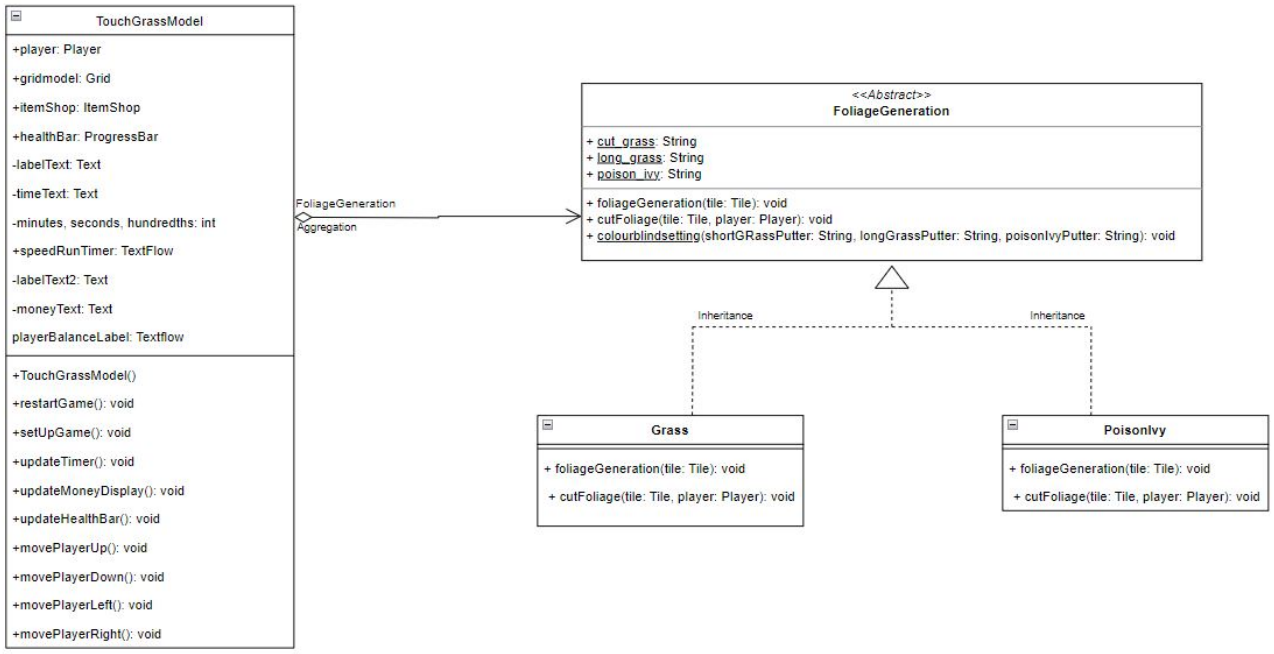
- LongGrass can be cut by the player and provides them money.
- CutGrass replaces LongGrass when it is cut, it also provides no money.
- PoisonIvy is unique such that it provides the largest sum of money.

These are further developed in TouchGrassView where we replace the sprites for other assessibility features.

Design Pattern #2: Strategy

UML

Nik - Strategy



Implementation Details:

The UML diagram uses the 'Strategy' design pattern and outlines these main components:

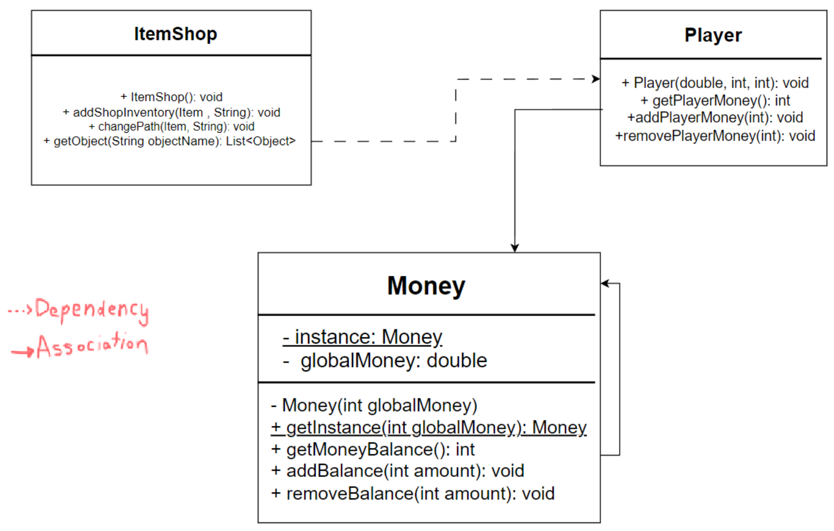
- Grass and PoisonIvy classes that inherit from the FoliageGeneration abstract class.
- The TouchGrassModel to switch between different FoliageGeneration strategies at runtime.
- The FoliageGeneration abstract class with a foliageGeneration() method and cutFoliage() method to be overridden by different generation strategies, depending on the type of foliage.

The Grass foliageGeneration() method takes in a Tile as a parameter, which ensures that the method accurately updates the correct tile with the new type of foliage. Similarly, the PoisonIvy foliageGeneration() method takes in a Tile as a parameter so that if poison ivy needs to be generated on a tile, it can accurately do so. Both methods and classes inherit from the FoliageGeneration abstract class and override the foliageGeneration() method when needed. The cutfFoliage() method is also an abstract method that gets overridden by both foliage types. For both Grass and Poison Ivy, it takes in a Tile, and Player object, and updates the status of the tile to be "cut". The FoliageGeneration abstract class receives its context from the TouchGrassModel, which holds all relevant data, including the gridmodel, which holds all the tiles, with each tile being a Hashmap with the keys being Integers, and the values being Tile objects.

Design Pattern #3: Singleton

UML

Youssef - Singleton


ItemShop
+ ItemShop(): void
+ addShopInventory(Item , String): void
+ changePath(Item, String): void
+ getObject(String objectName): List<Object>
Player
+ Player(double, int, int): void
+ getPlayerMoney(): int
+addPlayerMoney(int): void
+removePlayerMoney(int): void
Money
- instance: Money
- globalMoney: double
- Money(int globalMoney)
+ getInstance(int globalMoney): Money
+ getMoneyBalance(): int
+ addBalance(int amount): void
+ removeBalance(int amount): void
Dependency
Association

Implementation Details:

Here are the implementation details for this diagram:

This UML diagram utilizes the Singleton design pattern for managing a global currency within the game. The Singleton pattern ensures that only a single instance of the Money class is created which provides a consistent point of reference for the parts of the program – the Player and Shop - that interact with the game's currency. The Player and Shop classes depend on the Money class for all transactions, maintaining data integrity and consistency. The single instance of Money prevents conflicts that could arise from having multiple instances of a global currency in different parts of the game.

Money Class:

- instance: A private static attribute that holds the single instance of the Money class.
- globalMoney: A private instance attribute that stores the total amount of money (of type double) available globally within the game.
- Money(): A private constructor that initializes the Money instance with the provided amount of money. It is private to prevent direct instantiation from outside the class.
- getInstance(): A public static method that returns the single instance of the Money class. Within the code implementation, If the instance does not exist, it is then created with the provided initial global money amount. Otherwise, the initial money amount is ignored, and the existing instance is returned.
- getMoneyBalance(): A public method that allows getting the global money balance.
- addBalance(): A public method that allows adding to the global money balance.
- removeBalance(): A public method that allows subtracting from the global money balance.

Interactions with Other Classes:

<u>Player Class:</u>

- A class containing methods such as the constructor, Player(), getPlayerMoney(), addPlayerMoney(), and removePlayerMoney() that interact with the Money class, allowing players to query their balance and add to their funds.

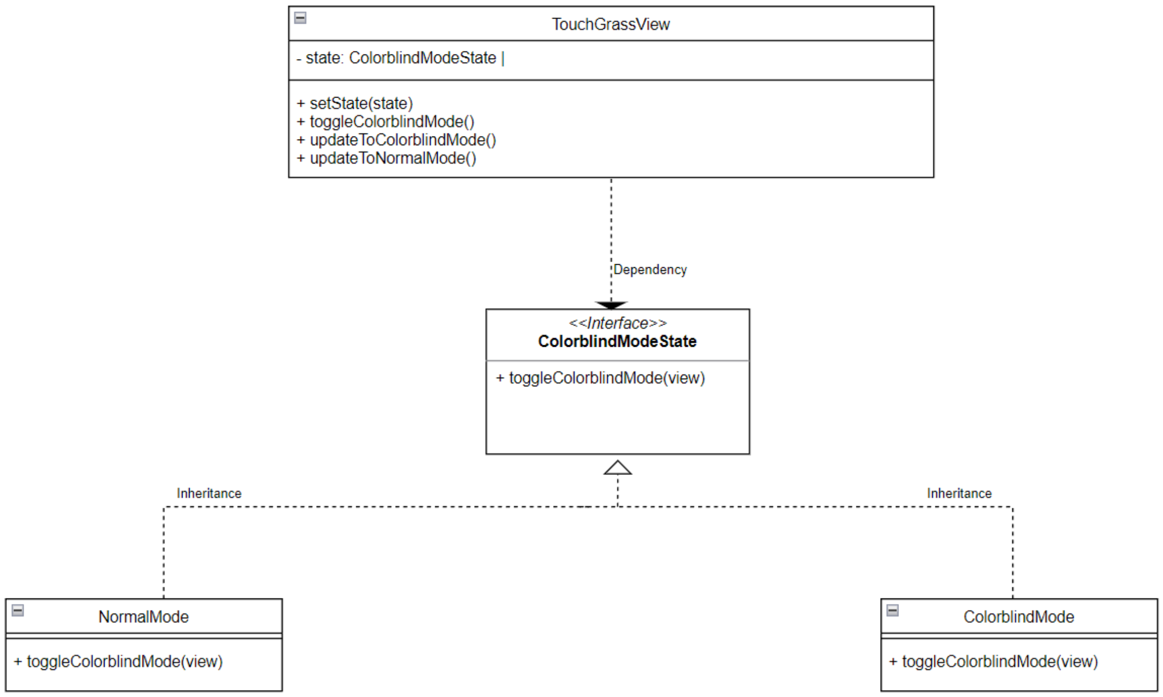
Shop Class:

- A class Contains methods like getMoney() and removeBalance() that directly interact with the player class and indirectly interact with the Money class, facilitating transactions where players can spend their money.

Design Pattern #4: State

UML

Adam -



Implementation Details:

The ColorblindModeState interface is pivotal in abstracting the state behavior in the TouchGrassView context. It declares a method to toggle the colorblind mode, toggleColorblindMode(TouchGrassView view).

Concrete states, NormalMode and ColorblindMode, implement this interface. They encapsulate the specific adjustments needed for their respective modes, such as modifying UI elements for color visibility.

TouchGrassView maintains a reference to the current ColorblindModeState. It delegates the action of switching modes to the current state's toggleColorblindMode method. It also provides updateToColorblindMode() and updateToNormalMode() methods for actual UI changes.

The UML diagram reflects a dependency from TouchGrassView to ColorblindModeState, signifying that TouchGrassView relies on this interface for managing state-related behavior. The inheritance from ColorblindModeState to NormalMode and ColorblindMode shows these concrete states are specific implementations of the interface.

This setup makes the application's state management flexible and extendable, allowing for future enhancements with minimal impact on existing code.

SECTION 4: ACCOMPLISHMENTS, LIMITATIONS AND TAKEAWAYS

In the end, our project was a success better than what we had imagined given our individual busyness. Although this doesn’t mean the project was smooth throughout. As the project carried on, we have switched responsibilities many times and covered for each other. The changes we had to make was for one of the following reasons:

- Code is more efficient
- Code is functioning properly through this method
- We’ve scrapped some things we initially wanted to implement
- We’ve added other features that have different requirements

These changes have been noticeable in our code, this final report, and our new UMLs. The major takeaway for us from this project is to always be our our toes and the importance of fluid communications. All in all, the final product we have ended up with has exceeded our expectations and we are pleased with how it turned out.