Object-Oriented Programming

Project, Phase 0

Kimia Fakheri

400101691

Question 1:How do you think the timeline can be implemented? How should this model be? Describe your proposed model by explaining the implementation method.

Answer:

The timeline can be implemented as a data structure such as an array or a list where each index represents a time unit. Each card's duration can be represented as a range in this array. When a card is played, its effect (damage or healing) is added to the corresponding indices in the timeline. The timeline marker can be an integer that increments after each round, executing the effects at the current index. This can be achieved by creating a 'Timeline' class with properties like 'marker'(the current position on the timeline) and 'effects' (a list of effects to be applied at each position on the timeline). The 'effects' property can be a list of lists, where each inner list represents the effects to be applied at a certain time unit. Each effect can be an object with properties like 'type' (heal or damage), 'amount', and 'target'(which player the effect applies to). When a card is played, create an effect object based on the card's properties and add it to the appropriate positions in the 'effects' list. The 'Timeline' class can have methods like 'moveMarker()' (which increments the marker and applies the effects at the current position) and 'addEffect()' (which adds an effect to the 'effects' list at the appropriate positions).

Question 2:Design 15 heal/damage cards and 5 spell cards for your game. And describe their characteristics and special abilities if any. (You can also use playing cards)

Answer:

Spell cards are as follows:

- Unagi: Reverses the next card of the opponent, causing damage to themselves, or healing their opponent instead.
- Akemi: Burns the next two damage cards of the opponent that come after it.
- Time Bomb: freezes the opponent for one turn
- Patronus: Shields the player from the next damage card
- Janus: A placeholder card that can be established after the opponent plays their last move.

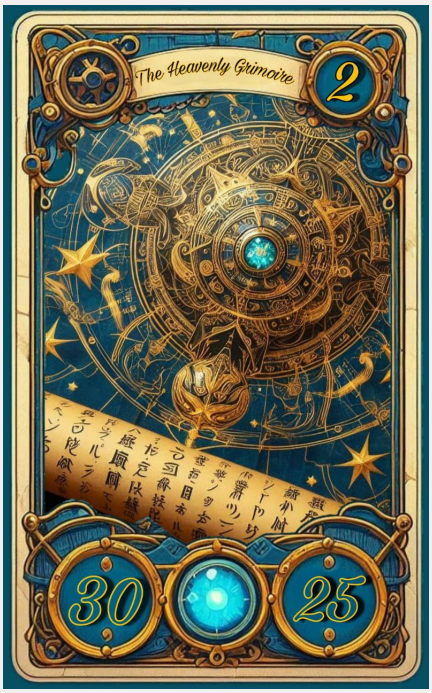

Figure 1: An example of Spell card.

Each defense/attack card has a number between 20 and 60 for damage/healing power and a number for health and Auray between 0 and 100. Also, each card has a number as uration and duration, and all these three numbers are present in the circles of the cards. The upper number of the card represents the duration of the card, the lower right number represents the power and the lower left represents the accuracy. Most of the attack/defense cards are simple, but some are marked with a star and are different from the rest, in that the attack card with the star is only healed by the healing card with the star, and the normal healing cards cannot remove its damage.
Here i have put 2 examples of Damage and healing card:

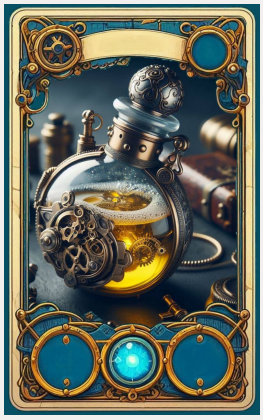

Figure 2: An example of Healing card.

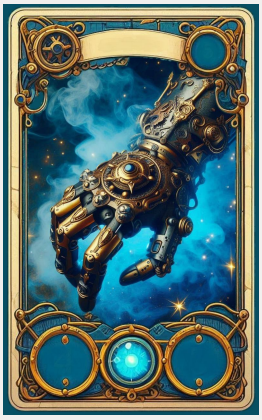

Figure 3: An example of Damage card.

Question 3:What classes and methods will you need to implement the gameplay and cards and how they inherit from each other. (Explain the general structure)

Answer:

I plan to have classes like ‘Player‘, ‘Card‘, ‘GameBoard‘, ‘Timeline‘. The ‘Card‘ class could have subclasses for different types of cards like ‘HealDamageCard‘ and ‘SpellCard‘. These subclasses would inherit from the ‘Card‘ class and could override methods to implement their unique behaviors. The ‘Player‘ class represents a player in the game. It should have properties like ‘name‘, ‘deck‘ (a list of ‘Card‘ objects), ‘hand‘ (a subset of ‘deck‘), ‘field‘ (a list of ‘Card‘ objects currently in play), and ‘totalDamage‘. It should have methods like ‘drawCard()‘, ‘playCard()‘, and ‘receiveDamage()‘. The ‘GameBoard‘ class represents the game board. It should have properties like ‘fields‘ (a 2D array representing the two parts of the board), ‘timeline‘ (an instance of the ‘Timeline‘ class), and methods like ‘destroyHouse()‘. The ‘Timeline‘ class represents the timeline. It should have properties like ‘marker‘ (the current position on the timeline) and ‘effects‘ (a list of effects to be applied at each position on the timeline). It should have methods like ‘moveMarker()‘ and ‘applyEffects()‘.

Question 4:Explain how to store the information of cards and players in the database. (Description of upgrade cards)

Answer:

I intend to use a relational database with tables for ‘Players‘, ‘Cards‘, and ‘Games‘. The ‘Players‘ table can store player information such as ‘playerID‘, ‘username‘, ‘passwordHash‘, ‘email‘, ‘totalWins‘, ‘totalLosses‘, and ‘currentHP‘. The ‘Cards‘ table can store card information such as ‘cardID‘, ‘name‘, ‘type‘ (heal/damage or spell), ‘attackDefensePoints‘, ‘duration‘, ‘playerDamage‘, and ‘specialAbility‘. The ‘Games‘ table can store information about each game played, such as ‘gameID‘, ‘player1ID‘, ‘player2ID‘, ‘winnerID‘, ‘loserID‘, ‘startTime‘, and ‘endTime‘. To handle card upgrades, I plan to have an ‘Upgrades‘ table that stores the upgrade levels for each card for each player. This table could have columns like ‘upgradeID‘, ‘playerID‘, ‘cardID‘, and ‘upgradeLevel‘.

Question 5:How can players upgrade their cards? What will be the reward of the winner and the penalty of the loser after each game?

Answer:

I propose that players can upgrade their cards using points earned from winning games. The winner could receive a certain number of points, while the loser might lose some points. The cost to upgrade a card could increase with each upgrade level. Each player could have a 'points' property that represents their current number of points. This property could be updated after each game based on the result (win or loss). The 'Card' class could have an 'upgrade()' method that increases the card's 'attackDefensePoints' and 'playerDamage' properties and decreases its 'duration' property. This method could also decrease the player's 'points' property by the cost of the upgrade. The cost of an upgrade could be a function of the card's current upgrade level. For example, the cost could be '10 * upgradeLevel', so the cost increases as the card is upgraded more times. After each game, the 'Game' class could call a method like 'awardPoints()' to update the players' points based on the result of the game. It could then call the 'upgrade()' method for the winner's cards.

Question 6:Considering that the game you will design will be for two players, unlike the original game, how do you design the general model of authentication and creating user accounts for two users? What classes and methods will you need to do this and how do they inherit from each other? (Explain the general structure, no details are needed)

Answer:

I suggest having a 'Player' class with methods for creating an account, logging in, and logging out. This class could use a 'PlayerDatabase' class to interact with a database table storing user account information. The 'Player' class represents a player of the game. It should have properties like 'username', 'password', 'email'(if needed!), and other variables that mostly concern in-game features. It should have methods like 'createPlayer()', 'login()', 'logout()'. The 'PlayerDatabase' class represents the database of player accounts. It should have methods like 'addPlayer()', 'removePlayer()', 'getPlayer()'. The 'createAccount()' method should take a username, password, and email as input. It should hash the password, create a new 'Player' object with the given username and password, and add the new user to the 'PlayerDatabase'. The 'login()' method should take a username and password as input. The 'createPlayer()' method should create a new 'Player' object.

Question 7:If you want to create a single player game, what method and algorithm will you use to design the opponent's player. Explain your method.

Answer:

For a single-player game, I plan to design an AI opponent using a decision-making algorithm like Minimax. The AI could evaluate the possible outcomes of playing each card in its hand and choose the one that maximizes its potential score. This can be achieved by defining a 'score()' function for the 'GameBoard' class. This function should return a numerical score representing the current state of the game from the AI's perspective. For example, it could return the AI's total damage minus the player's total damage. Next, implement the Minimax algorithm in a method of the 'AIPlayer' class (which inherits from 'Player'). This method, which you could call 'chooseCard()', should do the following: For each card in the AI's hand, create a copy of the current game board and simulate playing that card. Use the 'score()' function to calculate the score of the game board after playing each card. If it's the AI's turn, choose the card that results in the highest score. If it's the player's turn, assume that the player will choose the card that results in the lowest score. To look ahead more than one turn, use recursion: the 'chooseCard()' method should call itself to simulate the next turn before calculating the score of each card. Finally, in the main game loop, call the 'AIPlayer''s 'chooseCard()' method whenever it's the AI's turn to play a card. This is a basic implementation of the Minimax algorithm. For a more advanced AI, we could use techniques like alpha-beta pruning to make the algorithm more efficient.

Links:

Here it is our Repositories links:

-My GitHub link.
-Me and my teammate GitHub link.
-Me and my teammate Trello link.