MILESTONE 3

GROUP 7: Julien Ouellette, Sobechi Madueke, Danae Morrison

TABLE OF CONTENTS:

Game Design Documentation

Name: Sheepy Time
Duration: 30-45 minutes
Players: 1-4 players

Game Objective:
Be the first to reach your pillow using your winks (head) on the scoreboard by strategically jumping the fence, catching winks, avoiding nightmares, and utilizing dream tiles!

Milestone 3 Decisions and Deviations from Milestone 2

As we learned more things in class related to design patterns, we found our approach to designing the game naturally followed other paths to achieve the same goals. With some of these decisions came potential trade-offs with the SOLID principles we learned, but we hope to satisfy the idea of making our program flexible to adaptation- if not with this milestone, then with the next. We also decided to forego including a resting phase and use of DreamTiles in this Milestone in order to reliably complete a run of the game with the members at our disposal.

Sobechi served as the manager and the tester. Julien and Danae served as developers, and work was shared between members for the design document and requested updated materials. Responsibilities were adjusted throughout Milestone 3 to adapt to the availability, strengths, and weaknesses of the members of our group. Some tests may be for previous versions of classes whose new versions could not be updated before the 6 pm deadline.

Card Classes

In Milestone 3, we did away with the Card interface, SleepCard and NightmareCard classes that would implement that interface, and all of the classes that would make use of those implementations to make collections of cards based on the number of players and specific kinds of nightmares. Instead, we have made one Card class using the builder design pattern. Another class, DeckGenerator, generates the deck for a specific game instance based on the number of players and the specific nightmare chosen. For now, it is designed only to facilitate one player with any choice of nightmare. This deck is stored in a Deck object. All cards are viewable through the use of the CardViewer class. Currently, we're using CardViewer in such as way that a new one has to be made for each card that the system wants to display, but this can be changed later so that its display method can take in a card and a player and display its information. We chose to design it this way instead to avoid issues with inheritance and the Liskov Substitution principle where the same behavior does not occur across methods (such as the way nightmares move). The CardPlayer class is responsible for carrying out the actions of a card as chosen by a player (or when the game plays a nightmare card) and updating different model classes to contain this information.

Card is a model class which stores the state of a card (which doesn't change). Deck is a model class (which changes as cards get taken from the deck, and they will eventually get added back when we implement a feature to refill a deck). CardViewer is a viewer class that displays information about a card to users. DeckGenerator is a controller class that interprets information given by the user through the interface (how the user interacts with the game) to generate the deck. CardPlayer is a controller class that carries out the actions of a card.

DreamTile Classes

In Milestone 3, we've created an MVC system much like we did with the Card classes described above to handle multiple aspects of the DreamTiles. It is currently not implemented to work with the running of the game, but the groundwork has been laid for adding it in Milestone 4. So there's a viewer class for displaying the rules of a DreamTile, model classes for storing information of a DreamTile (like associated Ztokens) and storing all the DreamTiles of the game in a collection to be chosen from, and a controller class that carries out the conditions of a specific DreamTile if an eligible player chooses to use an ability, updating the system to match the use of the ability.

Board Class

There isn't just one "board" that interacts with other classes as we had in Milestone 2- all players, the nightmare, and dream tiles have a board assigned to them which contains their location. These are model classes. There is one viewer class, BoardViewer, that will display all of the information together to the user. For both this system and the others created to follow MVC, we will have to make sure that there is no interaction between the viewer classes and the model classes. Currently, our controllers such as the Initializer, RacingPhase, and RestingPhase (and others) should handle this, but improvements can probably be made after this first implementation is complete and feedback is received.

Score Class

The score class does what our Scoreboard class laid out separately for each player in Milestone 2, but does not contain the rules to update the scores of each player. This is so that the score class can remain a model and another class, a controller, can be in charge of updating the scores of each player as appropriate at the end of a racing phase.

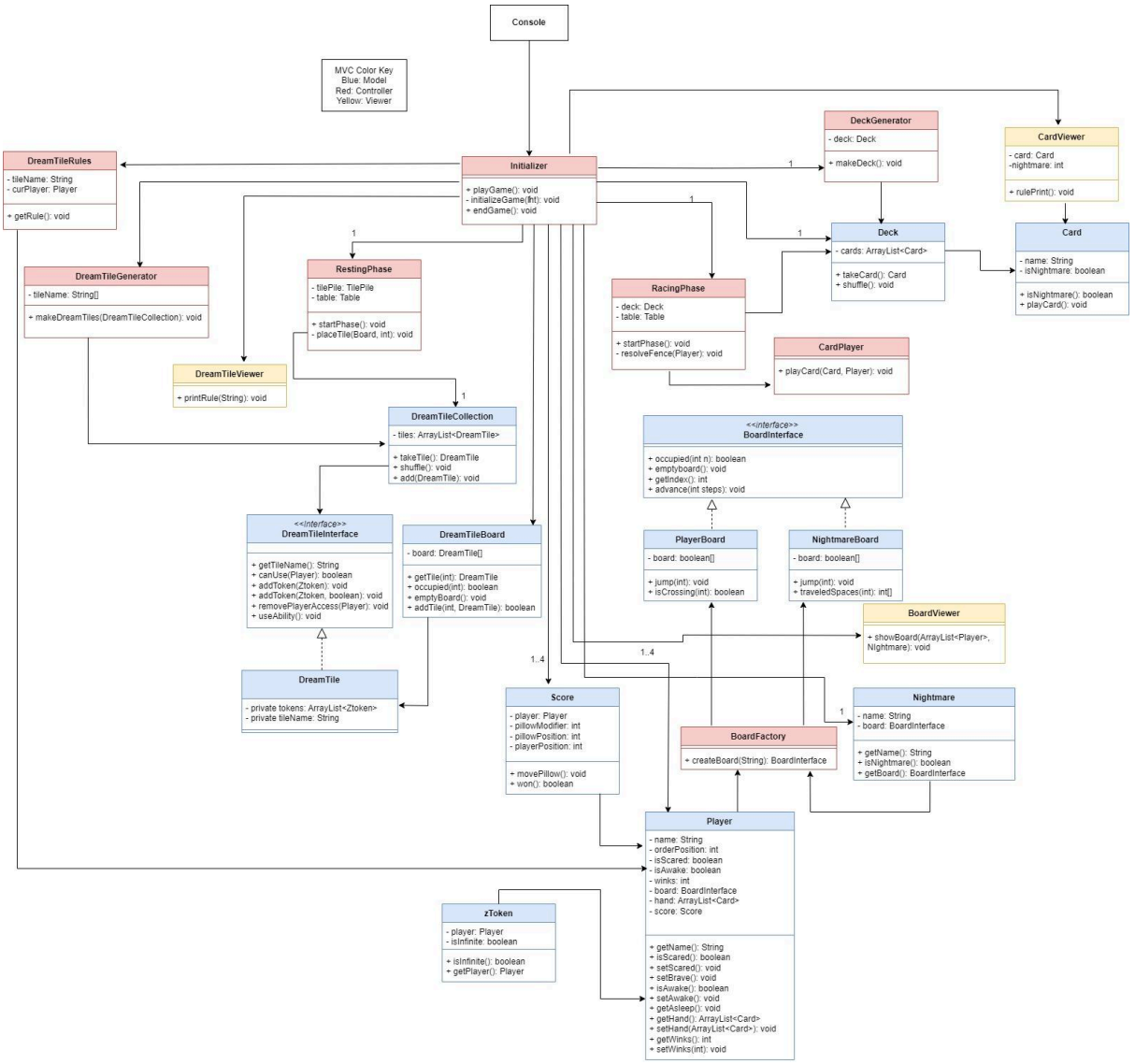
Sequence Diagrams

The sequence diagrams were made to cover what we saw to be the more important cases that could come about when playing the game. As we got into implementing the game, it became easier to see what the important cases would be. We designed the diagrams at a time that might not cover what the implementation currently does, but it follows the basic logic that we stuck to with our Milestone 3 implementation.

UML Diagram & Sequence Diagrams

The UML Diagram and the sequence diagrams for this milestone were designed to represent the program before finalized details were made during the period of testing and rewriting. Therefore, some details might not match directly with the classes and approaches that are taken to achieve various goals, but the diagrams below capture the spirit of our program well. The UML class diagram is color coded to specify the classes that belong to the MVC components. The color keys are Blue = Model, Red = Controller, and Yellow = Viewer.

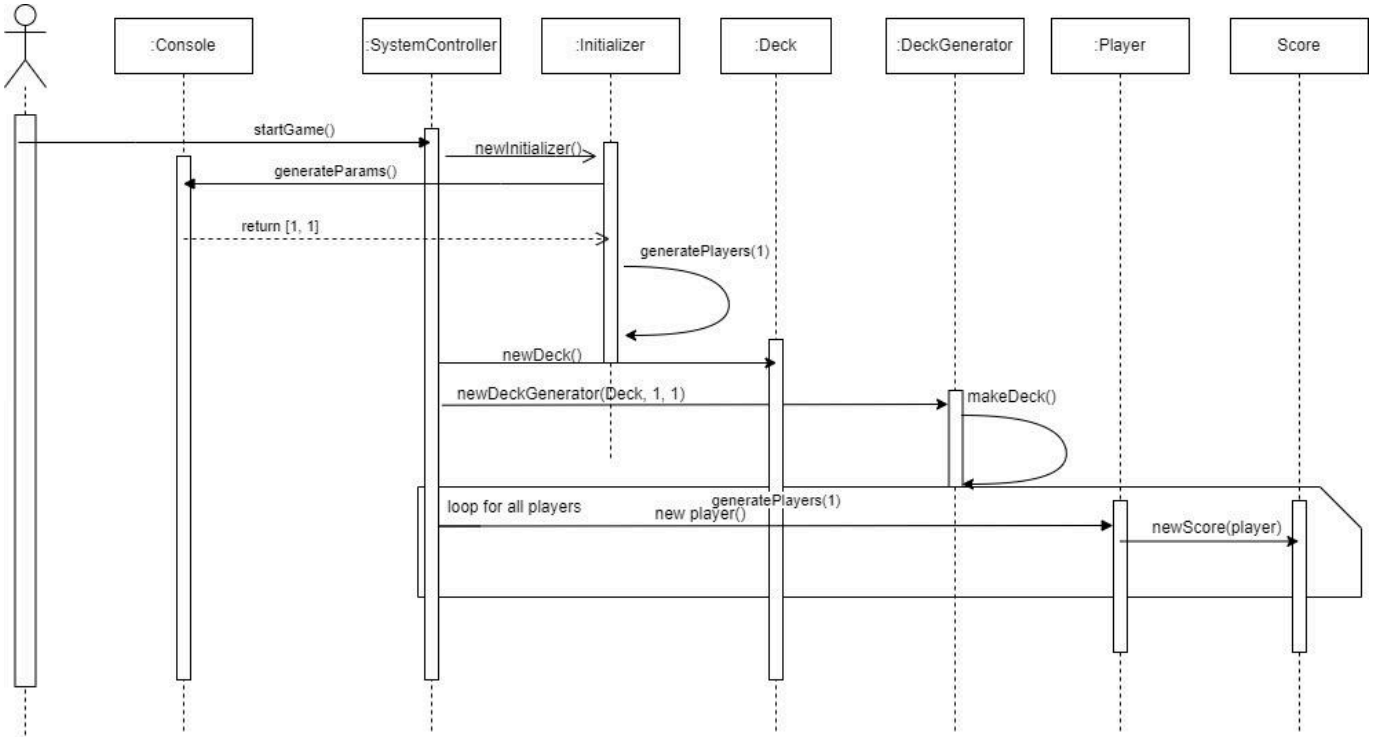
UML Class Diagram



Sequence Diagrams

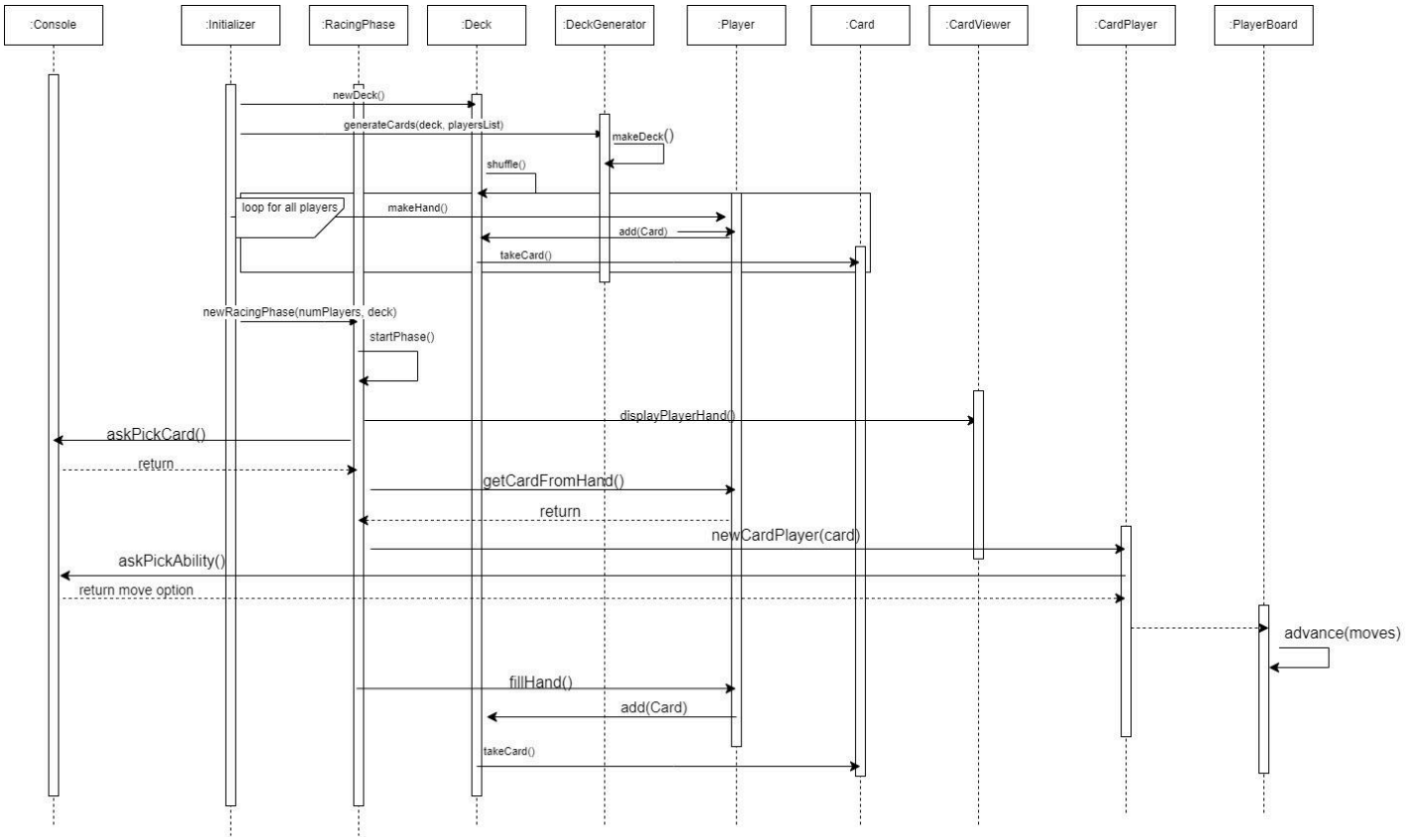
Start Game Solo Player With Wolf Nightmare

This sequence diagram covers the scenario of initializing a game for a user who is playing a game by themselves against the system with a wolf nightmare. The game is initialized for a solo player with a wolf nightmare, with a deck created by the DeckGenerator, players, and the score objects being created to match the given scenario.



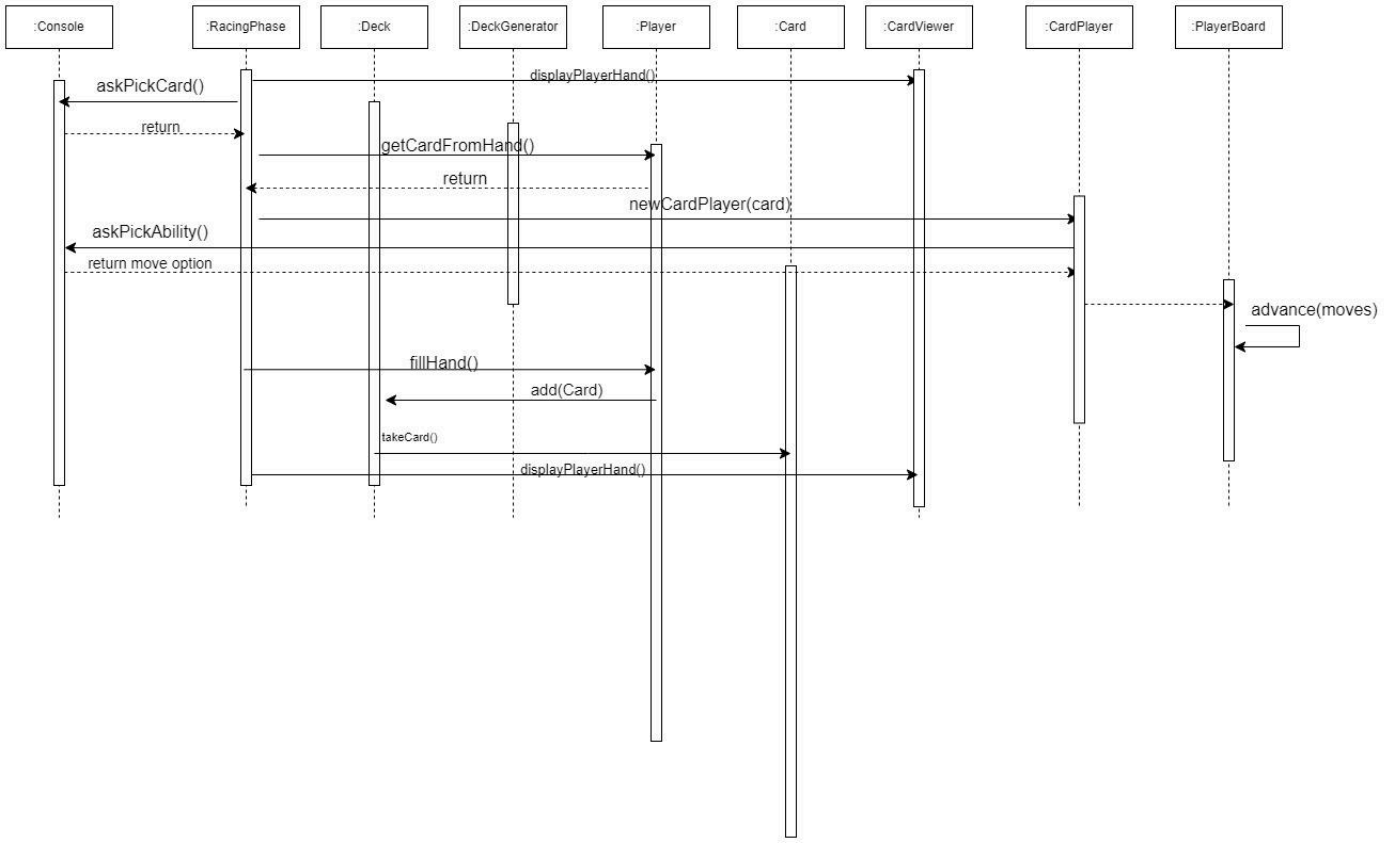
First Turn In Racing Phase and Moving

This sequence diagram covers the scenario of filling a player's hand with cards at the beginning of a racing phase and taking the player through picking a card from their hand, picking an ability (moving, in this case) from the card, having the system update all model classes to store this new information, and add a card to the player's hand so that they have two cards again.



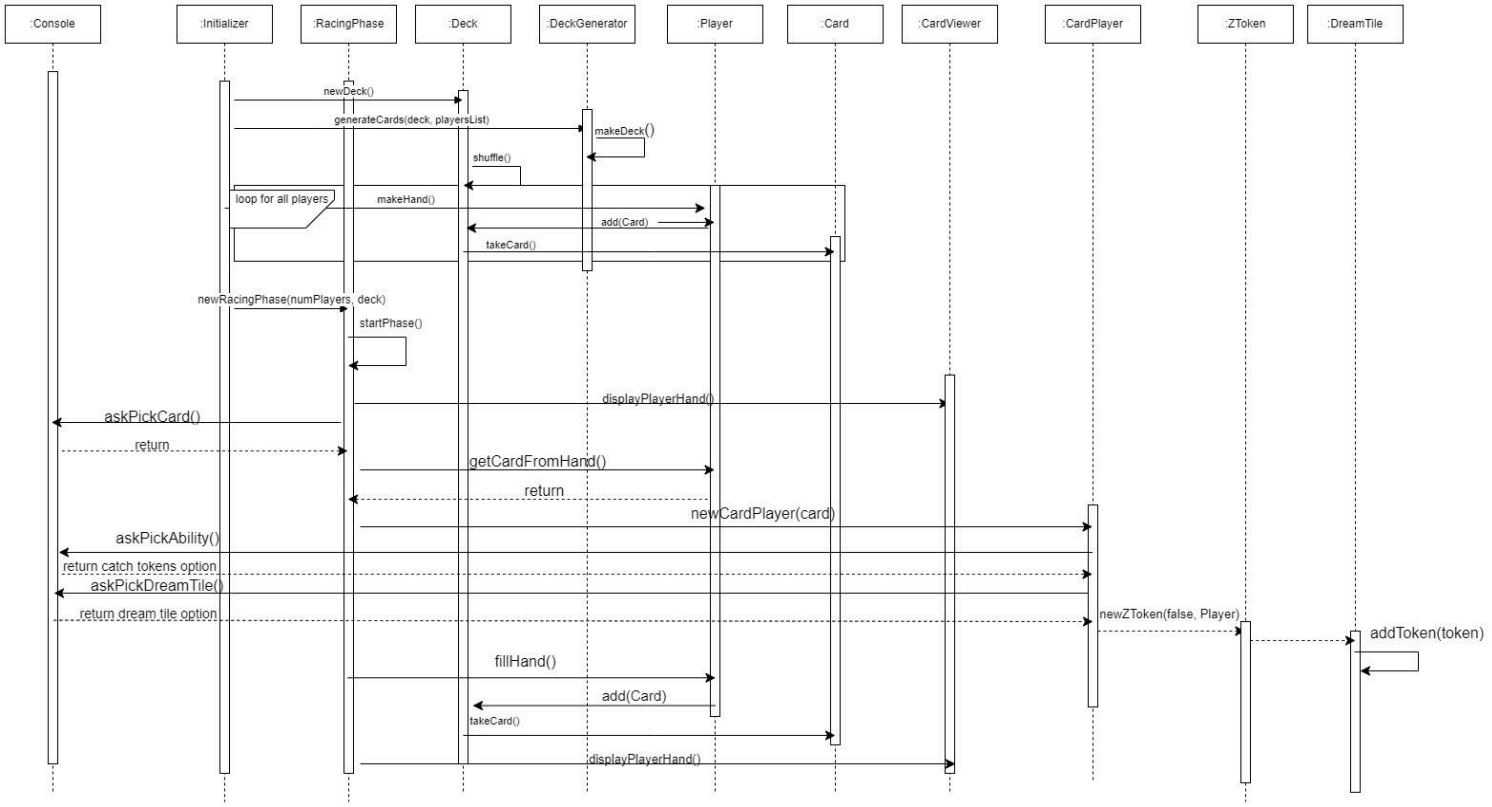
Player Moves in Racing Phase

This sequence diagram covers the sequence for when a player takes a regular turn (that is, not their first turn) in a racing phase and decides to move their player by a number of spaces.



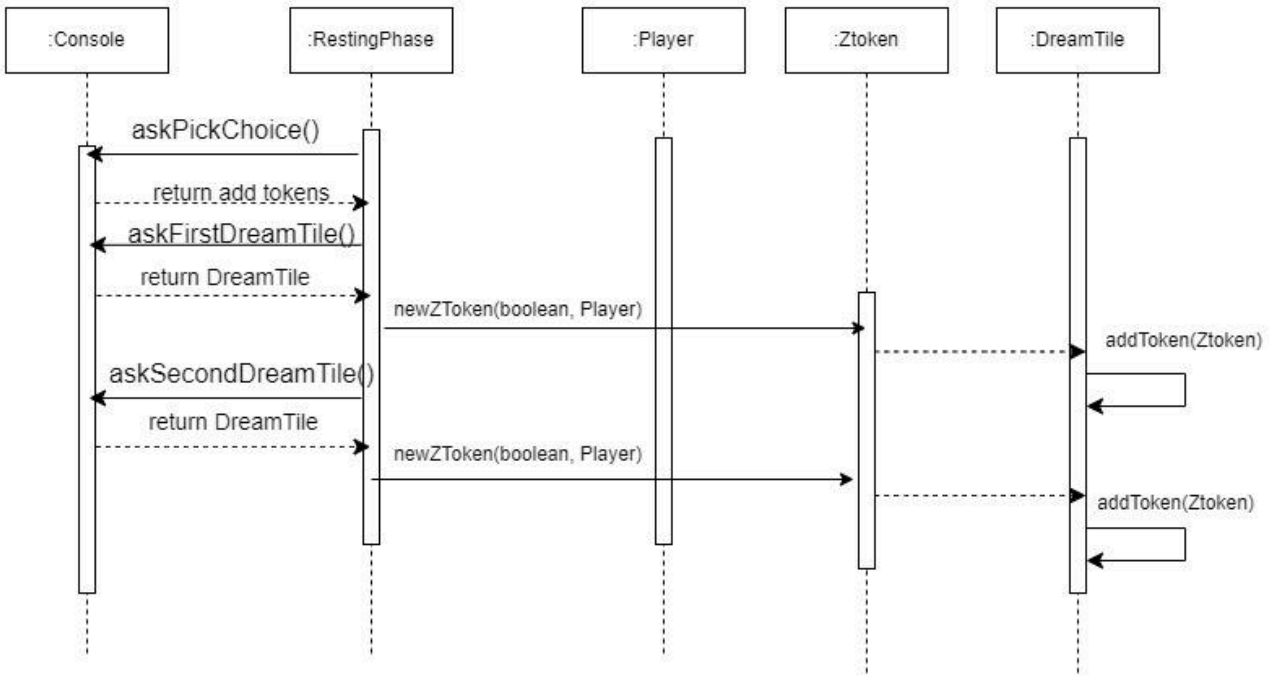
First Turn In Racing Phase and Catching Tokens

This sequence diagram covers the scenario of a player's first turn in a racing phase where they decide to catch tokens. It includes filling the player's hand with cards and taking them through the process of making their choice of card and ability to be used, including where to place their tokens.



Player Places Tokens in Resting Turn

This sequence diagram covers the scenario of when a player chooses to place two tokens on existing dream tiles during their turn in the resting phase. The system asks the player to choose which tiles they would like to place the tiles on and updates models accordingly to store the updated information.



User Journeys

Setting up the game:

1. The system asks the users how many players there are, between 1 and 4
2. The user responds with a number of players between 1 and 4.
3. The system asks the user to pick a difficulty from 1-3, 1 being the easiest and 3 the hardest
4. The user responds with a difficulty between 1 and 3
5. The system creates the game deck with the corresponding sleep and nightmare cards. The corresponding scoreboard and the corresponding nightmare are displayed.

To fill a player's hand with two cards at the start of a game (racing phase):

1. The game draws cards from the deck until the player's hand is filled with two cards.

Completing a player's turn (racing phase):

1. The player chooses one of two sheep cards in hand displayed on screen
2. The action/s on the card are resolved
3. The game draws cards from the deck until the player's hand is filled with two cards.
4. The player's turn ends

Calling it a night (racing phase):

1. The system moves the player to a position that brings them past the fence
2. The system prompts the player to decide if they want to keep playing

3. The player decides not to play anymore for the round
4. The system updates the state of the game to remove the player from the board and the playing order of active players

Using a dream tile ability(racing phase):

1. During a player's turn in the racing phase, the player lands on a position on the board that is associated with a dream tile and at least one of their tokens on it.
2. The system prompts the player with a question of whether or not they want to use the ability
3. The player responds in the affirmative and the system carries out the actions of the dream tile.
4. The system resolves the Zzz token usage
5. The rest of the racing phase ensues

Completing a player's turn (resting phase):

1. The player chooses to add a new dream tile to the board or place 2 Zzz tokens on existing dream tiles
2. The system resolves the choice of the player
3. The player's turn ends

Use Case Specifications

Use Cases

ID:	GameStart100Solo
Title:	Start Game of One Player
Description:	• This use case has the goal of initiating the game for a user who is playing by themselves. This use case is always required in order to accomplish other goals related to playing the game as a solo player.
Primary Actor:	The would-be player of the game
Preconditions:	• The state of the game is empty. It's waiting on the would-be player to supply information.
Postconditions:	• The system has initiated all the appropriate classes to correspond to the beginning of a game before the beginning of the first racing phase.
Main Success Scenario:	1. The game asks the user how many players there will be, between 1 and 4 2. The user responds with 1 3. The game asks the user to pick a difficulty from 1-3, 1 being the easiest and 3 the hardest 4. The user responds with the desired nightmare via a difficulty between 1 and 3 5. The system uses the information to create an appropriate deck, scoreboard, player objects, and other important details to prepare the game for a solo player entering the racing phase.

Extensions:	1. If a user enters a number of players that's not between 1-4, then they will be asked again to enter the number of players that will be playing 2. If a user enters an inappropriate response for nightmare type, they will be asked again to enter their desired nightmare
Frequency of Use:	Once, at the beginning of a game where there will be one player
Status:	Yet to be developed
Owner:	Yet to be decided
Priority:	Very important to the game's development

ID:	MyFirstStepsRacingTurn

Title:	Player Moves In First Racing Turn
Description:	• This use case has the goal of filling a player's hand during their first turn in the racing phase and getting the player through their complete turn in the phase by moving on the board.
Primary Actor:	One of the players of the game
Preconditions:	• The game has been initiated and the current player has not had played a turn yet this round
Postconditions:	• Racing phase is ongoing- that is, at least one player is still active (so they will have two cards)
Main Success Scenario:	1. The game draws a sleep card from the top of the deck, displays it, and adds it to the player's hand 2. The game draws another sleep card from the top of the deck, displays it, and adds it to the player's hand. 3. The player chooses one of their cards to play 4. User chooses the option to move their character on the board 5. The system carries out the chosen action and updates the state of the game to show the movement, which is displayed to the user. The card is added to the used cards pile 6. The system draws a sleep card from the top of the deck. It is displayed to the player and then put into the user's hand. 7. The player's turn ends
Extensions:	1. If the system pulls a nightmare card from the deck, it will carry out the actions of the card immediately and that card is added to the used pile of cards. The system will keep drawing cards for the user's hand until it is filled with two sleep cards 2. If the player lands on a position with a nightmare on it, they get scared.

Frequency of Use:	• Once per every player in a game per racing phase. So, with 2 players, for example, this can happen twice in one racing phase
Status:	Not yet developed
Owner:	Yet to be decided
Priority:	Very important to the structure of the game

ID:	PlanningAheadFirstRacingTurn
Title:	Player Catches Token In First Racing Turn
Description:	• This use case has the goal of filling a player's hand during their first turn in the racing phase and getting the player through their complete turn in the phase by associating one of their tokens to a dream tile.
Primary Actor:	One of the players of the game
Preconditions:	• The game has been initiated and the current player has not had played a turn yet this round
Postconditions:	• Racing phase is ongoing- that is, at least one player is still active (so they will have two cards)
Main Success Scenario:	1. The game draws a sleep card from the top of the deck, displays it, and adds it to the player's hand 2. The game draws another sleep card from the top of the deck, displays it, and adds it to the player's hand. 3. The player chooses one of their cards to play 4. User chooses the option to catch a token and place it on a dream tile 5. The system carries out the chosen action and updates the state of the game to show the token placement, which is displayed to the user 6. The system draws a sleep card from the top of the deck. It is displayed to the player and then put into the user's hand. 7. The player's turn ends.

Extensions:	1. If the system pulls a nightmare card from the deck, it will carry out the actions of the card immediately and that card is added to the used pile of cards. The system will keep drawing cards for the user's hand until it is filled with two sleep cards
Frequency of Use:	• Once per every player in a game per round. So with 2 players, for example, this can happen twice in one racing phase
Status:	Not yet developed
Owner:	Yet to be decided
Priority:	Very important to the structure of the game

ID:	MyMovingRacingTurn
Title:	Player Moves in Racing Turn
Description:	This use case has the goal of getting a user through a turn in the racing phase.
Primary Actor:	• One of the players of the game that is asleep (that is, active, even if scared)
Preconditions:	The game has been initiated and at least one player is still asleep
Postconditions:	• Racing phase can still be ongoing- that is, at least one player is still active (so they will have two cards)- or the racing turn could be complete
Main Success Scenario:	1. An active player chooses one of their cards to play 2. User chooses the option to move their character on the board 3. The system carries out the chosen action and updates the state of the game, which is displayed to the user 4. The system draws a card from the deck. It is displayed to the user and then put into the user's hand. 5. The player's turn ends

Extensions:	1. If the system pulls a nightmare card from the deck, it will carry out the actions of the card immediately and that card is added to the used pile of cards. The system will keep drawing cards for the user's hand until it is filled with two sleep cards 2. If the player lands on a position with a nightmare on it, they get scared. 3. The player can use a dream tile if their character lands on it and they have a token on that tile. If they choose to do so, then the game carries out the actions of the dream tile 4. If the player crosses the fence, the game informs the player and updates their scoreboard with the added 5 winks. The system prompts the player to decide if they want to keep playing and resolves the player's response.
Frequency of Use:	• Very often. As long as the players are in the racing phase then this will occur all the time
Status:	• Not yet developed
Owner:	Yet to be decided
Priority:	Very important to the structure of the game

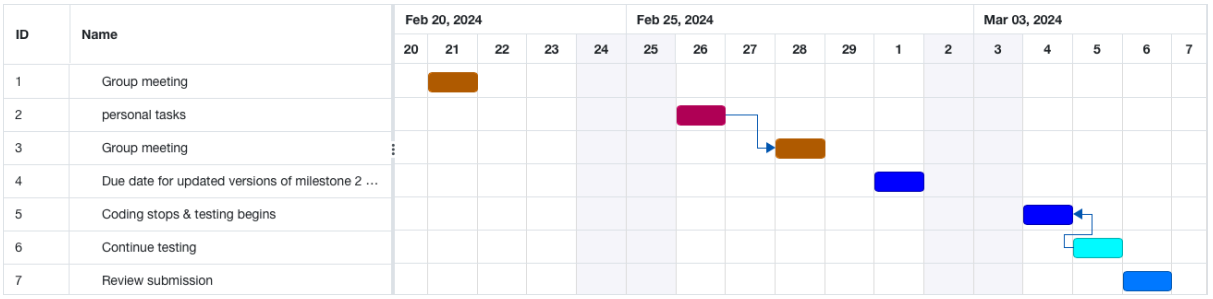
ID:	MyRestingTurnPlacesTokens
Title:	Player Places Tokens During Resting Turn
Description:	• The goal of this use case is to get a player through their turn in the resting phase where they choose to place two tokens across existing dream tiles.
Primary Actor:	A player of the game that is awake
Preconditions:	No players are on the board. The player's character is awake.
Postconditions:	• A new dream tile is added to the board with some (or one) of the player's tokens OR two of their tokens are added to existing dream tiles
Main Success Scenario:	1. The player chooses to add two of their tokens to dream tiles already attached to the board 2. The system asks the player where they would like to place the first token 3. The player responds with a tile/position and the system updates the state of the game to store and display this 4. The system asks the player where they would like to place the next token 5. The player responds with a tile/position and the system updates the state of the game to store and display this 6. The player's turn ends

Extensions:	1. If all positions on the board have a dream tile attached to it, then the user can only add 2 of their tokens to pre-existing dream tiles 2. If the user doesn't have 2 available tokens, the system alerts them of this and either their last token is placed on a chosen tile or they have to choose a dream tile to add to the board, as long as there is an available position to place the dream tile on
Frequency of Use:	• Once for every player after a racing phase when no one has won the game yet.
Status:	Not yet developed
Owner:	Not yet decided
Priority:	Important to the structure of the game

Addressing feedback from milestone 2

1. Adhered to the Class renaming suggestions from milestone 2 in this milestone (i.e, DreamTilePile is now DreamTileCollection).
2. We specified what classes fell among the MVC components using color keys. The color keys are: Blue: Model, Red: controller, Yellow: Viewer.
3. We provided a title and brief description for each sequence diagram and aimed for more correspondence between the sequence diagrams and the use case specifications.
4. We modified our user journey to specify the goal each is trying to achieve.
5. We included extra use case specifications.
6. Using a Collection module, we have implemented a shuffle mechanism for cards and dream tiles.
7. We've included more justification of the MVC components and decisions we made in general.

Gantt Chart



Deviations from schedule:

- This was originally designed to adhere to the original due date which was March 6th. Due to the changes there was deviation to the schedule above.
- We held the group meetings as scheduled and also communicated frequently via text.
- The coding implementation went up until Wednesday March 12th as developers decided to expand on more ideas (using design patterns taught in class) to clean up the code that was originally meant to be submitted on March 6th.
- Testing began as said for the classes developers agreed were ready to go.
- Testing went on till March 13th.