CRC Cards

The initial design of our snake game

Important classes and keywords

1. Snake: A snake object class that can be controlled by the player. The Snake is the basic class. A player can only interact with one snake object. The Snake object will respond to the keys pressed by the player (WASD or ↑↓←→). The Snake will immediately turn to the direction indicated by the player. The snake also has other attributes. One is the isDead. If the head of the snake collided with the body of another player controlled snake or the edge of the room, isDead becomes true the snake will send the data to the server saying that the snake is dead. The other attribute is the length of the snake. When a snake eats a food (A class inherits from the Item class), the snake will grow by one block and it will immediately be shown on the screen. When a snake eats a Potion, its body length will increase by 10 blocks. When a snake eats a Poison, its body length will shrink by 10 blocks. (If the snake has less than 10 blocks, then the snake will just be 1 block length).

2. Item: A item object class that can be used by the player who is controlling the snake. The item objects are the items that are in the server room that will be generated in the ServerRoom. All items have a location and will be consumed by the snake. The item will be consumed by the snake if the head of the snake reaches the coordinates. There will be 3 classes that inherits from the parent class: Item class. The first one will be food. The second one is the Poison, when a snake eats it, it will shrink by 10 blocks. The last one is the Potion, when a snake eats it, its body length will increase by 10 blocks.

3. ServerRoom: A server room that can hold a number of snake players. The SnakeGameModel after being completed for one player can be stripped off and changed into a network. This will count up to 4 players and send information into a new updated SnakeGameModel that will simply be listed.

4. Multiplayer: When we get to this point we will have to split the game representation of a player into an array list of 4 players and handle each set of information based on indexing. Eventually it will be the same exact logic but done 4 times over instead of once to update all 4 players. If there are less than 4 players, a null will be held and every calculation will check the player for null status, if null from either death or not joining nothing will update.

5. MainMenu: A main menu for the snake game. I am thinking At first will be an object to just start the game, then when networking we can add a room for up to 4 people to join. If there is any additional time we might be able to leave room for a speed and turn radius setting before the start of the game.

6. ScoreBoard: Probably a class in itself that the SnakeGameController will hold. Might be best to make this algorithm based on a very simple one based on snake parts, and other players beaten by cutoff. The View can use this to show the player's scores. The model can update the information as it happens.

Abstractions

JavaFX mvc

Snake:

1. SnakeGameView : extends application and uses animation timer to update the visual game based on information contained in the controller. It will have Pane as the root, and an arraylist of GameAssets which will represent the sections of the snake. This will also take the players input, if the button is pushed it will update the controller so the model can reflect the change.

2. SnakeGameController: Controller that will use calculated information from the model and update itself to the status of the players snake, its state in the game (alive or dead), its current state in movement (boolean, left,right, or if both are false strait), and and arraylist of Strings that can be broken down with split() that will represent a snake objects sections by positions rotation and sprite.

3. SnakeGameModel: A fully functional javafx app that will handle the calculations for positions and collision. During its animation updates It will use in-game ticks to calculate the next position of the player, and update each piece of the player to be one tick behind. It is dependent on the Snake class for calculations

4. SnakeHead: A class to represent the main movement of the player, It contains a node as a visual representation and a list of nodes as its tail pieces. After colliding with a food object, it will self maintain its own linked list of tailpieces, add them to the root, then adjust the ordering so the newest piece will take the place as the head (so it will layer in front of the tail). Inherits GameAssets

5. SnakeTail: A class representing the tail pieces of the snake that will be the visual representation of the user. The most important aspect is each piece will have a reference to its parent peice (the next piece in line), and on an update will update its x y and rotation to be one tick behind its parent peice. Inherits GameAssets.

6. Snake: an interface containing the SnakeHead and SnakeTail class. It can be inherited by the model, view, or server to give the classes full access to either of the two inner classes and their functionality as a fully implemented snake object.

7. Wall: A gameAsset that will just be a stationary rectangular representation of a wall. In the case of this game if collided with it will end the current player's game. Inherits GameAssets.

8. GameAsset: an abstract class that will have some of the basic principles all assets will require. A node root, which can be constructed into any shape requires, x , y, and rotate positioning, to the object to be manipulated around the pan; and will all have an update method, which when called will update the Node in the pain to be in the correct position. Key for polymorphism to simplify the act of treating objects as game objects.

9. FoodItem: An object that will randomly appear on the board and if collided with will be removed and add a new SnakeTail to the SnakeHead object. Inherits GameAssets.

Class: Snake	
Responsibilities	**Collaborators**
• Represents a single snake object that will be controlled by the player • Head will turn according to user's control (via pressing keyboard) • Will record if the snake is dead • Record the current length of the snake body	• SnakeTail • SnakeGameView • SnakeGameController • SnakeGameModel •

Class: Item	
Responsibilities	**Collaborators**
• A GameAsset that if collided with will disappear and add to the player who ate its tail. • Will randomly appear, any player can eat a item, and there WILL be a limit to how many can appear at a once	• SnakeHead Subclasses: • Poison • Food • Potion

Class: Poison	
Responsibilities	**Collaborators**
• Represents the visible piece of the poison • Inherited from the class Item • When a snake eats it, the snake shrink by 10 blocks • If the snake is less than 10 blocks, then the snake will be just 1 block.	• SnakeHead

Class: Food	
Responsibilities	**Collaborators**
• Represents the visible piece of the food • Inherited from the class Item • When a snake eats it, the body length will increase by 1 block	• SnakeHead • Snake

Class: Potion	
Responsibilities	**Collaborators**
• Represents the visible piece of the potion • Inherited from the class Item • When a snake eats it, the body length will immediately increased by 10 blocks	• SnakeHead • Snake

Class: SnakeHead	
Responsibilities	**Collaborators**
• Represents the actual location of the user. • Keeps track of all of its own pieces in a linked list • Holds a Node object that will updated in a pane and represent the players current x y and rotate values • The model will function and act as the source for every player to updated from • The view will add a node then change the positioning based on the information held by the controller	• SnakeTail • SnakeGameView • SnakeGameModel

Class: SnakeTail	
Responsibilities	**Collaborators**
• Represents the actual visible piece of the snake • Keeps track of its own and its parents positions and always positions iself one tick behind its parent • After a certain point will be able to collide with the head object ending the game	• SnakeHead

Class: SnakeGameController	
Responsibilities	**Collaborators**
• Retaining and updating the games information. • Communicated with the model to keep its information updated. •	• SnakeGameModel • SnakeGameView

Class: SnakeGameModel	
Responsibilities	**Collaborators**
• Using an animation timer to update the state of the game • Keeping track of a player's actions, state, and collision detection. •	• SnakeGameController

Class: SnakeGameView	
Responsibilities	**Collaborators**
• Displaying the state of the game to the user • Allowing the user to input keystrokes to control his character	• SnakeGameController

Class: SnakeGameServer	
Responsibilities	**Collaborators**
• An upgraded model that will run and send out information packets (String) that will updated all players on the potsition and state of all players. • Much like the original model, will completely function within itself using the animation timer to calculate movement and collision.	• SnakeGameListener

Class: SnakeGameListener	
Responsibilities	**Collaborators**
• A replacement for the SnakeGameModel, that will communicate with a server and update its own information based on incoming data • The controller will use this new information for the state of the game	• SnakeGameController • SnakeGameServer

Class: ScoreBoard	
Responsibilities	**Collaborators**
• **A class that will contain an algorithm to measure and calculate the score for a player.** • **Can either be a static utility class that can be called when needed, or a instantiated class in some sort of list or array to represent each player** • **Scores can be displayed by the view** • **Can write the current high score to a file to be retrieved or replaced when needed, and displayer for a solo or multiplayer player to beat.**	• **SnakeGameController** • **SnakeGameView**

Class: Wall	
Responsibilities	**Collaborators**
• **A Simple stationary game asset that will end the game if collided with**	• **SnakeHead**

Class: Food	
Responsibilities	**Collaborators**
• **A GameAsset that if collided with will disappear and add to the player who ate its tail.** • **Will randomly appear, any player can eat one, and there WILL be a limit to how many can appear at a once**	• **SnakeHead**