Key game functionalities of

(i) set up the initial game board (including randomised positioning for dragon cards): in the UML the use of the GamePanel class and AssetSetter class it sets the initial game board which includes the placement of dragon cards (objects) on the game panel. Further It shuffles a list of valid coordinates and places dragon cards (spiders, bats, eggs, and lizards) on the board according to specific rules.

(ii) flipping of dragon ("chit") cards;

- The Cards class represents the dragon cards in the game.
- It provides methods to set the front and back images of the card, flip the card, and check if the card is flipped.
- Mouse click events are detected in the GamePanel class, and upon clicking a card, its flip status is toggled.

(iii) movement of dragon tokens based on their current position as well as the last flipped dragon card;

- The move(steps: int) method allows the player to move a certain number of steps on the game board.
- In GameManager methods like changeTurn(), movePlayer(player: Player, steps: int), and processFlippedCard(card: DragonCards) handle player turns, movement, and processing of flipped cards.

(iv) change of turn to the next player

- Again in GameManager the methods will help indicate when to change to the next player

(v) winning the game.
In GameManager:

- checkWinningCondition() method checks if any player has met the winning condition.
- The displayWinningMessage() method could display a message or take action when a player wins.

In Player class:

- Winning conditions could be checked through the checkWinningCondition() method, which returns true if the player has won.
- The game manager utilizes this method to determine if any player has won the game.

are covered in your design

Design Rationales

1. Explain two key classes (not interfaces) that you have included in your design and provide your reasons why you decided to create these classes. Why was it not appropriate for them to be methods?

- Cards Class:
    - Complexity and State Management: Playing cards have several attributes and states associated with them, such as their position (x, y), size, front and back images, flip state. These attributes need to be maintained and manipulated throughout the game. A class provides a neat encapsulation of these attributes along with methods to manipulate them, making it easier to manage the complexity.
    - Reusability and Extensibility: By encapsulating card behavior within a class, it is reusable across different parts of the application. For example, the different cards that were formed (ie bat card, spider card), I am able to instantiate multiple Cards objects with different properties and images. Additionally, it's easier to extend the functionality of cards especially if we were to add an extension to our game.
- TileManager Class:
    - Data Management: The TileManager class stores information about tile images (Tile objects) and their positions (mapTileNum). This data needs to persist throughout the game and be accessible by different classes. By encapsulating this data within a class, you ensure its integrity and provide methods to manipulate it.
    - Separation of Concerns and Modularity: Similar to the Cards class, the TileManager class ensures a clear separation of concerns by encapsulating tile-related functionality. This modularity makes your codebase more maintainable and allows for easier debugging and testing.

2. Explain two key relationships in your class diagram, for example, why is something an aggregation not a composition?
3. Explain your decisions around inheritance, why did you decide to use (or not use) it? Why is your decision justified in your design?
   Inheritance was used for the formation of the dragon cards for the game board. This was used to promote code reusability. Inheritance allows the OBJ_Bat, OBJ_Spider, OBJ_Egg, OBJ_Lizard class to inherit all the properties and methods of the DragonCards class. This means that the child classes have access to the image, name, x, y properties, as well as the draw method without having to redefine them. This promotes code reusability and avoids duplicating code. Also using inheritance can promote polymorphism, since OBJ_Bat, OBJ_Spider, OBJ_Egg, OBJ_Lizard extends DragonCards, instances of those classes can be treated as instances of DragonCards when necessary.
4. Explain how you arrived at two sets of cardinalities, for example, why 0..1 and why not 1…2?
   For the class Character and Player the cardinality is one to one (1..1). This was decided one since that for each player instance there will be a single set a Character attributes connected to it thus it wouldn't have been a one to many as there would be a unique set of Character attributes for that one player given that if there were more the user will get confused about which game character is which. On the other hand, the cardinality between GameManager and Player is one to one or more, given that there is only one Gamemanager it will manage all the players game movements and actions.

However, there can be one more player in each game thus resulting in the (1..*) cardinality.

5. If you have used any Design Patterns in your design, explain why and where you have applied them. If you have not used any Design Patterns, list at least 3 known Design Patterns and justify why they were not appropriate to be used in your design

   In the design the Deign Pattern Observer Pattern was used. In the GamePanel class it implements the MouseListener interface, which is a form of the Observer pattern since the mouse events are observed then the code will react to the events accordingly. In the code the methods mouseClicked, mousePressed, mouseReleased, mouseEntered, and mouseExited represent the observer methods that respond to mouse events which only mouseClicked is fully used since the code requires the click input from the user to select which card to flip over. Thus the code observes if this action is completed then once a card is clicked the code shall enable the card selected to be flipped over to show what is behind it.

   The code utilises iteration to traverse through collections of objects. For example, in the paintComponent method in the GamePanel, a loop is executed to iterate over the obj array to draw each object onto the screen. Similarly, the initializeCircles method iterates over a collection of available images to assign them to individual Cards objects.
   These iterations allow the program to access and process each element in the collection sequentially without exposing the underlying structure of the collection.

   Dependency Injection enables loose coupling between components or classes. This is displaed in the TileManager class, which the class receives an instance of GamePanel in its constructor. This allows TileManager to interact with the GamePanel and access its properties, such as tileSize and maxScreenCol. By instancing GamePanel into TileManager, it makes the TileManager more flexible and reusable. For example, in another sprint if the functionality of GamePanel is changed or extend in the future, this can be completed without modifying TileManager, promoting a maintainable code.

   Resource Acquisition Is Initialization (RAII) is used in the TileManager class as it handles the resource loading within its constructor. The tile images are assigned to Tile objects, and are loaded using ImageIO.read() from resource streams obtained through getClass().getResourceAsStream() and the map data is processed and stored in the mapTileNum array. This initialization step ensures that the resources are ready for use after the TileManager object is constructed and are properly acquired and initialized when an instance of TileManager is created.

Test Case:

Test ID	Description	Steps	Inputs	Expected Result	Pass/Fail
TC1	Verify that GamePanel is instantiated correctly.	1. Create a new GamePanel object.	None	GamePanel is instantiated with the correct dimensions and background color.	Pass
TC2	Verify that the game setup is performed correctly.	1. Call the setupGame() method.	None	Game assets are set correctly.	Pass
TC3	Verify that the game thread starts successfully.	1. Call the startGameThread() method.	None	Game thread starts without errors.	Pass
TC4	Verify that the game panel is painted correctly.	1. Call the paintComponent() method.	None	Game panel is painted with the correct components and layout.	Pass
TC5	Verify that mouse clicks on cards are handled correctly.	1.open program 2. mouse click on card	Mouse click on a tile on the window	Cards flips	Pass
TC6	Verify that the images on eth card renders correctly and is displayed.	1. mouse click on card. 2. Mouse click on flipped card	Mouse click on shape (card)	Cards flips and reflips back. The images are displayed on the back of the card. The front of the card images is also rendered	Pass
TC7	Verify the card randomisation	1.open the game 2.click on a card 3.check the image of one of the cards 4.remember the card that was clicked on	Mouse click on card and reopen program	The flipped card will have a randomised image. Once the program is reopened	Pass

		5.close and reopen the program 6. click on same card 7. view the card		the same card that was clicked on previously will have a different image	
TC8	Verify the card remains the same throughout the game	1.open the program 2. click on a card 3. click on same card to flip back over 4. click on same card again	Mouse click on card	The card flipped will still be the same once flipped over and reflipped again – same game	Pass
TC9	Verify the randomisatio n of game board	1.open program 2. check the board 3. roughly remember the tiles on the gameboard (ie bat, spider) 4. reopen the program 5. view the newly formed game board	None	The game board will be unique for each game played, a randomised board will appear	Pass
TC1 0	Verify that TileManager is instantiated correctly.	1. Create a new TileManager object.	None	TileManager is instantiated with the correct parameters and initializes tile images and map tile numbers without errors.	Pass
TC1 1	Verify that tile images are loaded correctly.	1. Call the getTileImage( ) method.	None	Tile images are loaded without any IOExceptions .	Pass
TC1 2	Verify that the map is loaded correctly from a file.	1. Call the loadMap() method with a valid map file path.	Valid map file path	Map is loaded without any exceptions and assigns map tile numbers correctly.	pass

TC13	Verify that tiles are drawn correctly on the game panel.	1. Call the draw() method.	None	Tiles are drawn on the game panel according to the map tile numbers and tile images.	Pass
TC14	Verify that Cards correctly determines if given coordinates are within its boundaries	Call the contains() method with test coordinates	[10,4]	Returns True if the coordinates are within the cards boundaries, otherwise false.	Pass
TC15	Verify that the main method instantiates a new Window object.	1. Execute the main method.	None	A new instance of the Window class is created without errors. And a window is opened	Pass

//Instructions on how to run the file //
----
author: Annie Ho
studentId: 33156581

Purpose: Instructions on how to build and run the executable of this software prototype.
Target Platform(s): the executable is using Java, all platforms are able to run the file as long as an appropriate JDK is installed on the user's machine.

************ INSTRUCTIONS ************
##Download JDK
1. Install JDK21 on whichever machine that will be running the software.

   Linux: https://www.oracle.com/au/java/technologies/downloads/#java21
   macOS: https://www.oracle.com/au/java/technologies/downloads/#jdk21-mac
   Windows: https://www.oracle.com/au/java/technologies/downloads/#jdk21-windows

   (was used to download on Windows machine:
   https://download.oracle.com/java/21/latest/jdk-21_windows-x64_bin.exe)

2. Complete the installation of the Java JDK by following the steps of the installer

SDK Set up
3. Set up the SDK for the file by going to:
   "File -> Project Structure..." [shortcut "Ctrl + Alt + Shift + S"]
   Click on the "Project" Tab

4. Under project, click on the drop-down bar next to the sub-heading SDK
5. Click on "+ ADD SDK" then "JDK...", find the directory of the downloaded JDK and click
on the whole file to add the JDK to the project
6. Click on "Apply" then "OK" to finish the SDK set up

Build Executable
7. Set up the SDK for the file by going to:
   "File -> Project Structure..." [shortcut "Ctrl + Alt + Shift + S"]
   Click on the "Artifacts" Tab
8. On the top left of the window, a "+" symbol will be present, add a new artifact by clicking this
9. Click on the JAR title
10. Then click on the "From modules with dependenices..." title, a new window will appear

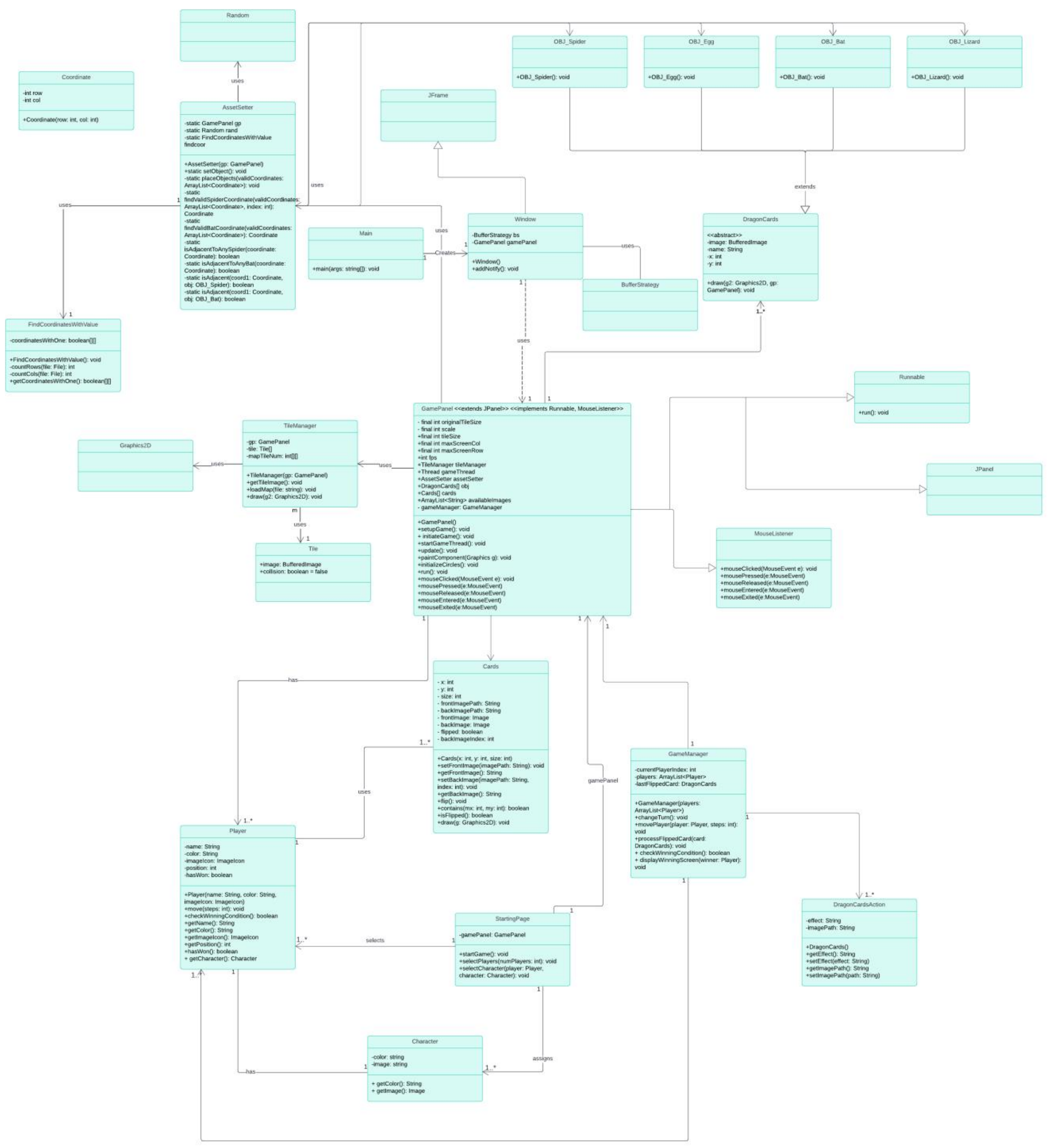
11. Next to the title "Main Class:" click on the folder icon and either search the main class
however it should appear on the window "Main (main)". Click ok after selecting the main class
12. Click on "Apply" then "OK" to finish making an artifact set up
13. After navigate to the "Build" tab on the top
14. Once it has been found, click on the tab and find the "Build Artifacts.." and click on that option
15. A pop-up will appear, and click build
16. An executable as been created it will be located in the out folder:

📦 fieryDragons
  └ 📁 out
    └ 📁 artifacts
      └ 📁 fieryDragonsGame_jar
        └ 📜 fieryDragonsGame.jar

Run Executable
Option 1:
17. Locate the jar file and double click to open the executable

Option 2:
18. Change directory in the terminal to where the jar file is being kept write in the terminal
"cd C:\Users\annie\IdeaProjects\CL_Monday06pm\out\artifacts\fieryDragonsGame_jar"
change based on where the user has saved the jar file
19. In the terminal write " java -jar fieryDragonsGame.jar"

YouTube Link:
https://youtu.be/XkfYCiehyDM


Random
uses
Coordinate
-int row
-int col
+Coordinate(row: int, col: int)
AssetSetter
-static GamePanel gp
-static Random rand
-static FindCoordinatesWithValue findcoor
+AssetSetter(gp: GamePanel)
+static setObject(): void
-static placeObjects(validCoordinates: ArrayList<Coordinate>): void
-static findValidSpiderCoordinate(validCoordinates: ArrayList<Coordinate>, index: int): Coordinate
-static findValidBatCoordinate(validCoordinates: ArrayList<Coordinate>): Coordinate
-static isAdjacentToAnySpider(coordinate: Coordinate): boolean
-static isAdjacentToAnyBat(coordinate: Coordinate): boolean
-static isAdjacent(coord1: Coordinate, obj: OBJ_Spider): boolean
-static isAdjacent(coord1: Coordinate, obj: OBJ_Bat): boolean
uses
1
1
FindCoordinatesWithValue
-coordinatesWithOne: boolean[][]
+FindCoordinatesWithValue(): void
-countRows(file: File): int
-countCols(file: File): int
+getCoordinatesWithOne(): boolean[][]
uses
OBJ_Spider
+OBJ_Spider(): void
OBJ_Egg
+OBJ_Egg(): void
OBJ_Bat
+OBJ_Bat(): void
OBJ_Lizard
+OBJ_Lizard(): void
JFrame
extends
Main
+main(args: string[]): void
uses
Creates
1
1
Window
-BufferStrategy bs
-GamePanel gamePanel
+Window()
+addNotify(): void
uses
BufferStrategy
DragonCards
<<abstract>>
-image: BufferedImage
-name: String
-x: int
-y: int
+draw(g2: Graphics2D, gp: GamePanel): void
1..*
uses
1
1
1
Runnable
+run(): void
JPanel
GamePanel <<extends JPanel>> <<implements Runnable, MouseListener>>
- final int originalTileSize
- final int scale
+final int tileSize
+final int maxScreenCol
+final int maxScreenRow
+int fps
+TileManager tileManager
+Thread gameThread
+AssetSetter assetSetter
+DragonCards[] obj
+Cards[] cards
+ArrayList<String> availableImages
- gameManager: GameManager
+GamePanel()
+setupGame(): void
+ initiateGame(): void
+startGameThread(): void
+update(): void
+paintComponent(Graphics g): void
+initializeCircles(): void
+run(): void
+mouseClicked(MouseEvent e): void
+mousePressed(e:MouseEvent)
+mouseReleased(e:MouseEvent)
+mouseEntered(e:MouseEvent)
+mouseExited(e:MouseEvent)
MouseListener
+mouseClicked(MouseEvent e): void
+mousePressed(e:MouseEvent)
+mouseReleased(e:MouseEvent)
+mouseEntered(e:MouseEvent)
+mouseExited(e:MouseEvent)
Graphics2D
uses
TileManager
-gp: GamePanel
-tile: Tile[]
-mapTileNum: int[][]
+TileManager(gp: GamePanel)
+getTileImage(): void
+loadMap(file: string): void
+draw(g2: Graphics2D): void
uses
m
uses
1
Tile
+image: BufferedImage
+collision: boolean = false
1
1
1
has
Cards
- x: int
- y: int
- size: int
- frontImagePath: String
- backImagePath: String
- frontImage: Image
- backImage: Image
- flipped: boolean
- backImageIndex: int
+Cards(x: int, y: int, size: int)
+setFrontImage(imagePath: String): void
+getFrontImage(): String
+setBackImage(imagePath: String, index: int): void
+getBackImage(): String
+flip(): void
+contains(mx: int, my: int): boolean
+isFlipped(): boolean
+draw(g: Graphics2D): void
1..*
uses
1..*
1
Player
-name: String
-color: String
-imageIcon: ImageIcon
-position: int
-hasWon: boolean
+Player(name: String, color: String, imageIcon: ImageIcon)
+move(steps: int): void
+checkWinningCondition(): boolean
+getName(): String
+getColor(): String
+getImageIcon(): ImageIcon
+getPosition(): int
+hasWon(): boolean
+ getCharacter(): Character
1..*
1
1..
gamePanel
1
GameManager
-currentPlayerIndex: int
-players: ArrayList<Player>
-lastFlippedCard: DragonCards
+GameManager(players: ArrayList<Player>)
+changeTurn(): void
+movePlayer(player: Player, steps: int): void
+processFlippedCard(card: DragonCards): void
+ checkWinningCondition(): boolean
+ displayWinningScreen(winner: Player): void
1
1
1..*
DragonCardsAction
-effect: String
-imagePath: String
+DragonCards()
+getEffect(): String
+setEffect(effect: String)
+getImagePath(): String
+setImagePath(path: String)
1
StartingPage
-gamePanel: GamePanel
+startGame(): void
+selectPlayers(numPlayers: int): void
+selectCharacter(player: Player, character: Character): void
selects
1
1
Character
-color: string
-image: string
+ getColor(): String
+ getImage(): Image
has
1
assigns
1..*