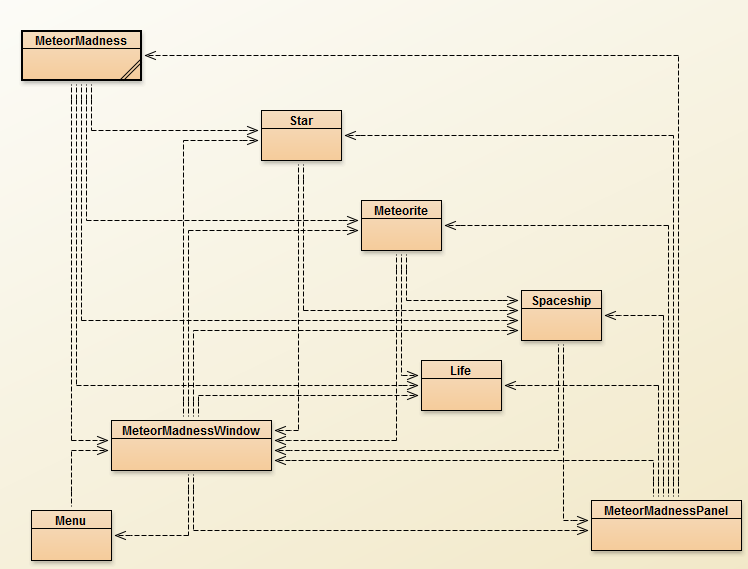
Design and Justification

The UML diagram of the project:



MeteorMadness Class

The MeteorMadness class holds many important responsibilities. The MeteorMadness class contains the main method which runs the game. It also contains a constructor which runs the actions of the objects in the game such as creating a new window with specified dimensions, running the game and adding the player (spaceship) and objects to the screen.

It has dependencies on the Spaceship, Star, Meteorite, Life and MeteorMadness Window classes from this project. Whenever the game runs, there needs to be a spaceship, falling stars, falling meteorites and lives and the fields of the class have been declared as private. Furthermore, in the class, isRunning is a private field of type Boolean and can be changed using getters and setters such as the methods getIsRunning() and setIsRunning(Boolean isRunning). This indicates that there is usage of encapsulation in this class.

Menu class

The Menu class allows users to start a new game by clicking on the JButton at the game’s menu page. It also stores the current highest score of the game and displays it by using a method that converts the score to a string. The Menu class also has methods that set the background of the menu interface as well as methods that display the title of the game, the start button and the current highest score.

The Menu class depends on the MeteorMadnessWindow and the MeteorMadnessPanel classes which extends JPanel hence inherits all the fields, methods and behaviour of JPanel. This allows the Menu class to implement the paintComponent(Graphics graphics). The paintComponent(Graphichs graphics) method is not a method within the class itself but is a method from JComponent which is inherited by JPanel and hence is inherited by Menu.

MeteorMadnessWindow class

MeteorMadnessWindow creates the window of the game and extends to Jframe which is a use of Inheritance that follows the basic principles of Object Oriented Programming. This allows for the addition of Panels such as the sub classes of JPanel Menu and MeteorMadnessPanel and allows all components inside the Panels to be viewed on the window. The createView() method creates the window that has the window’s title and button components such as the close or minimize functions. There is use of polymorphism in the add(…) and remove(…) methods. These methods add and remove sprites from the window. These are used in the main class MeteorMadness under the createGameLoop() method.

Another functionality is that the class implements KeyListener and ActionListener. The KeyListener determines the movements of the Spaceship in the MeteorMadnessPanel through the methods keyPressed() and keyReleased(). The keyPressed() method adds positive or negative speed to the Spaceship whenever the left and right arrow keys are pressed and the keyReleased sets the Spaceship’s speed to zero when the keys are released. The ActionListener listens to when the button on the menu is pressed and it’s functions are under the actionPerformed() method. This method plays the background music when the game starts and allows the window to switch from displaying the Menu Panel to the MeteorMadnessPanel Panel. The MeteorMadnessWindow depends on the following classes: Spaceship, Meteorite, Life, Menu, Star and MeteorMadnessPanel.

MeteorMadnessPanel class

MeteorMadnessPanel contains a collection of objects that are shown in the window. It also extends to JPanel which is a use of inheritance that follows the basic principles of Object Oriented Programming. The createView() method sets the dimension of the panel in alignment with the MeteorMadnessWindow. The refreshRate() method is set to trigger the repaint() function every 10 milliseconds. The paintComponent() method allows the graphics to be re-painted and is useful for animation in the game. It also uses a function called ‘synchronization’ which is like a thread that synchronizes the task and works between two lists. For example, the Spaceship, Meteorite, Star and Life classes run simultaneously as shown in the game but if all lives are lost the game ends and displays ‘Game Over’ with your high score stored.

Spaceship class

The Spaceship class represents a controllable spaceship. The image and coordinates of the spaceship is kept within this class. The class sets the parameters of the spaceship and creates a sprite of the spaceship with a location and velocity. The drawSpaceship() method draws the image of the spaceship by deciding its behaviours and prevents the Spaceship from going outside the boundary of the window by setting conditions to check the position of the spaceship.

The class is dependent upon the MeteorMadnessPanel and MeteorMadnessWindow classes. This is shown, for example, in the MeteorMadnessWindow class where the methods for the movement of the spaceship using the left and right arrows are found. The keyPressed() and keyReleased() methods control whether the spaceship is moving. When the left cursor key is released, the speed will be set to zero with the setSpeed() method and the spaceship will stop moving.

Life class

The Life class represents a set of hearts that shows the amount of lives the player has within the game. The class displays a set of 3 hearts that disappear when the Spaceship encounters a meteorite. It has no dependencies on any other classes within the project therefore making the class more universally applicable if the class were to be reused. The class displays 3 hearts that are decreased each time a meteor hits the spaceship and displays this to the user. The class contains the Life() constructor that imports the image of the hearts and set the height and width. The class uses getters and setters to set the position and dimension of the heart and sets the count for the players lives and returns this to the screen.

Meteorite class

The Meteorite class is responsible for falling objects (meteorites) that the Spaceship must avoid. An important functionality is that the meteorites must interact with the spaceship. Its position, dimension and speed at which it is falling is attributed. Public setters and getters are then used to set and get the values of attributes. This confirms encapsulation. The accessor and mutator methods for ‘speed’ field is as follows;

public double getSpeed(){return speed;}

public void setSpeed(int speed){this.speed=speed;}

The use of constructor creates Meteorite objects in the MeteorMadness class. The Meteorite class depends on the Spaceship class because they interact with each other. If a meteorite interact/collide with the Spaceship more than 3 times, then the game is over. Therefore, collision detection was added to the Meteorite class as a behaviour. It depends on MeteorMadnessWindow for positioning.

Star class

The Star class is responsible for falling objects (stars) that the spaceship needs to collect. One of its main functionalities is to interact with the spaceship. The falling stars will contribute to the score if collected by the spaceship. It has a position, dimension and a falling speed. Again, the use of accessor and mutator public methods ensures encapsulation.

The use of constructor allows the creation of star objects in the MeteorMadness class. Its dependency with the Spaceship class is because stars interact with the spaceship. If a star interacts/collides with the Spaceship, a certain score will be added. To execute this functionality, collision detection was added to the Star class as well. It depends on MeteorMadnessWindow for positioning.

Additional Functionalities

Added sound to play when the game opens this was done by taking advantage of the javax.sound library and implimenitng that in a method called playMenuSound() in the MeteorMaddnessWindow class.

Adds a Game Over string at the end of the game when all lives are lost by using if-else statement in the paintComponent() method in the MeteorMaddnessPanel class to draw the string on the screen.

How did the Team Operate?

The group sat together to decide who would be responsible for which classes after the main idea and classes where decided. Each member did research on their part and collaborated with another member to determine the best way to implement the code if their class had a dependency on the other.