Tile super class

For all classes that occupied a space in the map (Tower, Shrub, Grass, WizOrPath): I used an abstract Tile class. This use of inheritance allowed for code reusability since all tile subclasses have common attributes and methods that are written within the tile super class. It also uses abstraction which allows for the sub classes to emphasise their specialties and thus increase code readability, which allows for an easier time debugging and understanding the code. The tile class also had protected attributes. This made it so data could only be edited by subclasses or classes within the package. Protection minimises the chance that code is unintentionally edited which otherwise could be error prone. These features combined allow for high modularity potential and minimal negative interactions with the base game.

WizOrPath super class

Paths and wizard house classes had common properties. By treating wizard houses as paths for the path finding algorithm: this allowed for a reliable base case to start creating the path from. This class utilised protection, abstraction, and inheritance for the same reason as the tile class.

Interfaces

Interfaces were used to allow for polymorphism. Polymorphism was useful because it allows the reader to understand that each method implementation has a similar purpose, increasing readability. It also allows for code to be easily extended in future by implementing these various interfaces. They also reminded me to implement the methods needed, which allowed my vision for each class to remain consistent throughout development. The interfaces themselves were used over abstract classes because multiple interfaces with different methods could be implemented at once, unlike abstract class.

Encapsulation

All classes except super classes and the app class used private attributes. This allowed for the attributes to stay within the method, minimising the amount of “spaghetti code” which would be confusing to a reader. Getters were used to make it harder for other classes to write into unrelated objects, minimising bugs; setters were used sparingly for this reason.

Extension - Poison

My extension activates a poison spell which damages all monsters on the screen every second. The damage per second, mana price, and time it lasts are all determined by the config file. Default values exist in case the config file fails to address these fields. This was implemented because towers were weak against fast monsters as they could outrun projectiles. Smart use of poison allows for towers to play to their strength in high damage for strong, slow targets.

Extension - Tower/Upgrade cursor indicators

A greyscale image of the tower hovers with the cursor while in tower mode to indicate where it will be placed. This allows for the player to get an idea for how their tower will be placed before spending the mana, allowing for smarter gameplay decisions. Similarly, different shapes corresponding to different upgrade modes hover with the cursor. This minimises the number of accidental upgrades to be made by the player. Both implementations make the UI distinct from competing submissions.