1)

Monster.java	Monster.location()	10
Monster.java	Monster.location(Point)	11
Monster.java	Monster.ping(Model)	15
Monster.java	Monster.draw(Graphics, Point, Dimension)	30
Keys.java	Keys.keyTyped(KeyEvent)	17
Keys.java	Keys.keyPressed(KeyEvent)	18
Keys.java	Keys.keyReleased(KeyEvent)	22
Direction.java	Direction.right()	6
Direction.java	Direction.left()	7
Direction.java	Direction.up()	11
Direction.java	Direction.right()	12
Direction.java	Direction.unUp()	13
Direction.java	Direction.unRight()	14
Direction.java	Direction.up()	17
Direction.java	Direction.down()	18
Direction.java	Direction.unRight()	19
Direction.java	Direction.right()	22
Direction.java	Direction.down()	23
Direction.java	Direction.unDown()	24
Direction.java	Direction.unRight()	25
Direction.java	Direction.right()	28
Direction.java	Direction.left()	29
Direction.java	Direction.unDown()	30
Direction.java	Direction.down()	33
Direction.java	Direction.left()	34
Direction.java	Direction.unDown()	35
Direction.java	Direction.unLeft()	36
Direction.java	Direction.up()	39
Direction.java	Direction.down()	40
Direction.java	Direction.unLeft()	41
Direction.java	Direction.up()	44
Direction.java	Direction.left()	45
Direction.java	Direction.unUp()	46
Direction.java	Direction.unLeft()	47
Camera.java	Camera.location()	11
Camera.java	Camera.location(Point)	12
Camera.java	Camera.ping(Model)	14
Camera.java	Camera.draw(Graphics, Point, Dimension)	17
Sword.java	Sword.location()	14
Sword.java	Sword.ping(Model)	24
Sword.java	Sword.draw(Graphics, Point, Dimension)	33
Viewport.java	Viewport.paintComponent(Graphics)	16

Total Overrides: 42

Direction.java	Direction.unUp()	8

(After finishing task 1-7, this can be added.)

2)

A)

This is the state pattern. State is held in the private direction field inherited by both Camera and Sword. This state is updated by passing a functional interface to the object which then executes it to set the correct direction.

B)

This method is called inherited from the Keys class and in this case is called when pressing W. It will set the direction for the player to move upward when the key is pressed (via actionsPressed.put()) and then will unset it when the key is released (via actionsReleased.put()).

It sets the direction (as mentioned above) by calling the set() method in Camera that is inherited from the ControllableDirection class. This will then call apply() on that Functional<Direction, Direction> interface which will then set direction based on the current direction (methods in Direction enum differ based on state) by calling the correct overridden method.

3)

A)

Cells.forAll() is currently used to perform actions on other Cells that are within a specific range. (Done via visitor pattern.)

B)

We could use this to implement an AoE ability where the character performs an attack on Coord p and int range determines the size of that AoE attack. A fire spell for example that sets any of those cells on fire (given that our game is a top-down Minecraft-y type game, this would be the equivalent of a ghast fireball) or a potion effect that gives buffs/debuffs to friends/enemies within a certain range (the Minecraft equivalent being a splash potion).

4)

A) This implements the strategy pattern.

B) This pattern is implemented by having each 'strategy' being the different levels. We create a new Viewport that contains a phase level. The levels will all have a model that is supplied by the Phase.levelX methods in the Phase class. This model has a ping() method that will then act on the Entity objects it has. The setPhase function doesn't need to know what ping(), it just knows that ping is a strategy that it can call on every *tick* of the game.

5)

COULDN'T ANSWER

6)
In this case we are passing an anonymous inner class so that we can override the `windowClosed()` method. This allows us to change the behaviour such that it calls `closePhase.run()` instead of the standard behaviour.

If we just passed a lambda, we would not be able to override the method as java has no way of knowing what method to override.

7)
A) Currently there is an incorrectly named method override in the Up() state of the Direction enum.

B) `Direction unUn(){return None;}` becomes `@Override Direction unUp(){return None;}` and is located on line 8 of the Direction.java file.

C) Spell Checkers, Checkstyle, and Linting functionality for IDEs like intellij/eclipse will all help. The checkstyle plugins could point out that we have a potentially misnamed variable (based on it being unused and similar to other methods). The spellcheckers would point out that the function is misspelt (unUn makes no sense in English). Linters will give us a quick highlight to show that the method or variable is unused.