James Butler

UB: 18011279

Task 1:

The 2D Array tiles, of type TileType (only TileType values can be placed in the array) is created taking the pre-set size values GRID\_WIDTH and GRID\_HEIGHT to set the correct sizing of the array. Using nested for loops it works through every position in the for loop and sets the position to the value SPACE.

The program then works through the tiles array and at each position it uses the rng method to generate a random double value. It then compares the value of the double to the pre-defined black hole spawn chance. If the random double is smaller than the spawn chance it sets the current array position to BLACK\_HOLE. It then repeats this process but instead of comparing to the black hole spawn it compares to the inactive pulsar spawn chance. If the random double is less than this spawn value it sets the current position to PULSAR\_INACTIVE. Finally, it returns the value of the array tiles.

Task 2:

Before completing the spawnPlayer method I completed the getSpawns method, so it could be used to spawn the player. I first created an array list, spawns, of type point. Every time the method is called it first clears all the values in the array list. As array lists aren’t restricted in length if I didn’t do this each time the method was called it would add more and more to the array list instead of replacing the old values which may no longer be empty. I then used two for loops to work through every position in tiles, one at a time. If the value of the position was SPACE, integer x and y would be set to the search position in the for loop. The position (x,y) is then added to the next free space in the array list spawns. The array list is then returned so the values can be accessed later.

I then completed the code for the spawnPlayer method. I first set the integer r to a random position in the spawns array list. I got the random position by generating a random double value and then multiplying it by the length of the spawns array list to find a random position in the array list. I then used the .get method to get the contents of the position that was randomly generated. With this point value I then used the getX and getY methods to get the x and y coordinates of the point. A player was then created with the hull strength 10 and the spawn position (a, b).

Task 3:

I first created the array of Asteroid objects called asteroids giving it a length of 10. I used a for loop when creating the asteroids so every position in the array had an asteroid in it. I selected a random position in the spawns array list using the same method discussed in task 2. Again using the getX and getY methods I split the point in to separate x and y coordinates. I then instantiated the asteroid class which allowed me to create an asteroid with the generated x and y coordinates. This asteroid was then added to the asteroids array. Once all the 10 asteroids had been created the array asteroids was returned.

Task 4.

Before I created the player movement method I created a new method asteroidCheck. This checked if the player was currently in the same position as an asteroid. It used the getX and getY methods to retrieve the coordinates of the player. Using a for loop it worked through all the asteroids and for any that weren’t equal to null it would compare the players coordinates to that asteroids coordinates. It retrieved the asteroids coordinates using the getX and getY methods on asteroids[k] with k being the current asteroid. An if statement checked if the coordinates of the player and the asteroid were equal. If they were equal it would set the value of that asteroid to null, add one to points and then check if points was equal to 5. If points was equal to 5 the newLevel method was called and it returned true. If none of the asteroids were in the position of the player it would return false.

I then completed the player movement methods (describing the movePlayerLeft method however all methods are similar with different direction movement and checks). The players positions were retrieved using getX and getY, set to two variable and the value of the x variable was reduced by 1. If the x variable was less than 0, the player was placed in the position (GRID\_WIDTH – 1, y) which greatest x position with the same y value. This allowed for an affectively endless map. The position of the x and y variables was then checked in the tiles 2D array and if the position was equal to SPACE it would place the player in the position. The asteroidCheck method was called in the if statement forcing the check if there was an asteroid in the current position. If the position was not equal to SPACE it would not move.

Task 5:

The code for the activatePulsar method used two for loops to search through the 2D array tiles and if any of the positions equalled PULSAR\_INACTIVE it would set that position to PULSAR\_ACTIVE. The deactivatePulsar method was the same but checked for PULSAR\_ACTIVE and would set that position to PULSAR\_ACTIVE.

The code for the pulsarDamage method was more complex. The variables x and y were set to the players x and y coordinates -1 which were found using the getX and getY methods. I then set variables a and b to 3 which would be used later in for loops. If x or y were less than 0 they had 1 added (to them to avoid an array index error) if x<0 one would be taken from a and if y<0 one would be taken from b. If x+2 or y+2 was greater than their respective grid sizing -1, a or b respectively would have 1 taken from their value. Two for loops were created using the variables x, y, a and b to limit them. If the tile being searched was equal to PULSAR\_ACTIVE, it means that an active pulsar is in a tile next to the player. If there was an active pulsar next to the player would have their hull strength reduced by 2 + however many levels they had cleared, this made it harder after each level was cleared.

Task 6:

The moveAsteroids method would work through the following for each of the asteroids. It would select a random position in the spawns array. If the current asteroids value was not null it would use the getX and getY methods to get the asteroids position. It would then use the getMovementDirection method to find the movement direction of that asteroid. It would then add or take away from the x or y value depending on the movement direction. If the new position was outside of the tiles array the asteroid would be moved to the position found at the start of the task using the getX and getY methods to find the points coordinates. If the asteroid was trying to move on to a black hole or pulsar it would again move the asteroid to the new position using the same code as used above. If it was trying to move in to a valid position it would be allowed to move to the destination.

Task 7:

To complete the spawn aliens method, I first had to create the array aliens of type Alien. I decided that there should be three aliens so I set the length of the aliens array to three. I used a for loop that worked through three times to fill the array. It first chose a random point in the spawns array list using the code described in tasks 2 and 3. The getX and getY methods where then used to split the point in to x and y coordinates. I then instantiated the Alien class to create a new alien with a hull strength I set to five and the x and y coordinates I had taken from the spawns array list. This alien was then added to the aliens array and when the for loop had worked through three times it would return the aliens array.

I then completed the code for the moveAliens method. I first set a variable to the length of the aliens array using aliens.length to find it. I then worked through a for loop however many times the variable equalled. For each position if it did not equal null it would call the moveAlien(Alien a) method. The moveAlien(Alien a) method would then use the getX and getY methods to find the position of the alien. I then used the rng method to generate a number between 0 and 3 (this allowed for me to have one value for each possible direction the alien code be moved). (describing code for move right, all others are the same but with different movement and adjusted checks) if the random number is 0 and x is less than the width of the map, x is increased by 1 and the aliens position is set to the new (x, y) position. If x + 1 is greater than the width of the map the position is set to (0, y). I then checked if the alien was in the same position as the player. I used getX and getY to get the players position and compared these coordinates to the aliens. If they were the same the players hull strength was reduced by 2. I finally checked if the alien is in the same position as an asteroid by comparing their coordinates. If they are in the same position points was reduced by 1 (alien collecting asteroids reduces the players score) and then the asteroid was placed somewhere else randomly using the same code as used when spawning the asteroids.

Task 8:

To complete the newLevel code I had to call some methods and change a few values. Firstly I added one to the cleared score as they have finished a level, then reset the points score so they have to collect the asteroids again. I then called the generateLevel method as this would create a new random map. getSpawns, spawnAliens and spawnAsteroids were then called to add aliens and asteroids to the new level. Finally, the placePlayer method was called to place the player on the new map.

The placePlayer method chose a random position in the spawns array list in the method previously discussed and then placed the player in that position. Finally, it removed that position from the spawns array list.

Task 9.1:

The code that allowed aliens to collect asteroids was discussed in the task 7 write up which checked if the aliens and asteroids where in the same position and would reduce their score if they were.

Task 9.2:

The code for this was described in tasks 4 and 7 as both the player and aliens were allowed to wrap around to the other side of the grid by altering their position.