README file for essential computing exam

Mathias Larsen (70434)

November 12, 2021

1 Introduction

For essential computing we have made a range of different assignments and used different methods to solve the problem given. This program Enemies.java is a small game where a "player" is controlled by a user to survive against enemies. For a better understanding of the program an explanation of every class has been made below:

2 Classes

2.1 Main and App

Starting with Main.java, we see that its simple and the only function of this class is to call the class App. In the case of Enemies this does not make any functional difference, although if this were to be a 'mini-game' in some even bigger game Main.java would be the 'bigger games' that would then call the smaller game in it self. continuing to App. App contains the main loop of the game, where we simply call map.show_map() to print the visual part of the map for every tick of the game. Which leads us to map.game_tick(). This method will be explained further in section 2.2, but makes sure every object on the map moves every game iteration. App also has one big variable called *running*, which is a boolean that switches to false when either the Player object dies or every Enemy object dies. Finally outside of the while loop we show the map one last time with map.show_map() so that the user can see that he has died. There has also been sprinkled a few print-statements all over App to make the game more user-friendly.

2.2 Map

The Map class is the class that mainly keeps track of every Entity on the map. This class is what ties the Enemy, Obstacle and Player class together. To start off with it is important to address the variables Entity[][] and ArrayList¡Entity¿. The Entity[][] is a 2D array that can only contain: an object whose class impliments the interface Entity, or null. The variable ArrayList¡Entity¿ is simply a list containing every Entity in the game. Looking at the constructor of the Map we see the the method fill_boarder(). This method has two forloops that loop throught the full Entity[][] and sees if the x or y value are at the boarder of the map. If they are, then we create an instance of Obstacle on that position. Looking at the constructor again, we see place_player(). This method makes an instance of the player class in the middle of the Entity[][] and adds the player to the ArrayList¡Entity¿. The next two methods, place_enemies() and place_obstacles() from the constructor are fairly similar, they only vary in the class they place. place_enemies() contains a for-loop that calls place_an_enemy() for however many times as we want enemies. In the code the amount of enemies is put to 2.

place_an_enemy() pickes out a random x,y and checks, if that position on the Entity[][] is empty, if yes, place an instance of the Enemy class there and add an Enemy to the ArrayList¡Entity¿, if not, call place_an_enemy() again, meaning that the program repicks a new x,y variable. Finally show_map() and game_tick() as mentioned in section 2.1. show_map() has a double for-loop that writes either a white-space if the spot on the map is null, otherwise it will use the draw() method from the given Entity. Outside of that there are a few newlines and empty prints to make it nicer to look at. game_tick() contains a foreach loop that interates through the ArrayList¡Entity¿ and checks if the Entity is alive then run act() in that Entity.

2.3 interface Entity

This is the interface for every class in the game that implements Entity, and simply means that those classes will inherit the methods of this interface. The reason for an interface is so we can go through the ArrayList¡Entity¿ and make every Entity run its act() method. This makes sure that every Entity acts once and only once.

2.4 Obstacle

The Obstacle class implements from Entity meaning in short that it has to have the same methods as any Entity in the game. Obstacles in general are very static points and just need to occupy a space on the map. That is why the class is mainly empty.

2.5 Enemy

The class Enemy also implements Entity, so like the Obstacle class, has to have the same methods. Although the main methods in Enemy are act() and move(). Starting with Act(). Act() reads off the players position on the Map and figures out in what direction, compared to the given Enemy instance, the player is. These directions are then put into the move() method. The move() method starts by checking if the spot the Enemy wants to move to is empty, it simply updates the x,y values of the Enemy, otherwise, if the spot the Enemy wants to move to has a player, then kill the player and move to that spot and finally in any other case; meaning that the enemy hits something else, i.e. an obstacle, then kill the Enemy.

2.6 Player

Finally the Player class is the third and final class to implement Entity and has therefore a the same methods as every other Entity. Of the interesting methods in Player are: Act(), draw(), jump() and move(). Act() starts by reading an input from the terminal and inputs that into a switch-statement. This statement will then see if the terminal input is within the span of the cases, if yes, move in accordance, if not, break. Bringing us to move(), this simply takes two integers as directions and if the direction the player wants to move in is empty, then it moves the x,y of the player in those directions, both visually and in the Map Entity[][]. If the spot the player wants to move to is not empty, then kill the player. The draw() method is fairly simple. this just draws the player as a P if the player is alive and as an x if the player has died. Finally the jump() method tries to move the player to a random spot on the map. This method uses recurrsion and starts by picking one random coordinate set (x,y) and if they are empty, move the player there, if they are not empty call jump(), in other words pick a new set of coordinates.