**Project:** Platformer

CSC1054

100 points total

This project is over about 3 weeks. **You are responsible for pacing out the project yourself**. Waiting to the last minute will probably not go well.

Summary: This game is a simple “platformer” game. The player [black square above] can move left, right, and jump up. “gravity” will push the player down if the player is jumping in the air. The goal is to reach the “victory block” [grey square above]. The level itself is read from a text file and then displayed in the JPanel. For simplicity, there is only one JPanel and the JFrame.

Create a JPanel/JFrame of approximately 800 x 600. (Make it so 800 by 600 pixels are seen when

**Setup and object hierarchy**

drawing; you may adjust it after finishing the project later on). Once done, there should be about a 5-10 pixel black border around the outside and you will paint in the middle with the game’s data.



The class GameObject has four variables: (int) x position, (int) y position, and the Color of this game Object. All game objects are rectangular and do not rotate. The x and y positions indicate the middle of the rectangle. Other than a constructor, accessors, and mutators, GameObject has two methods:

* *collides*, which takes in another game object and determines if the two game objects collide. This method is given in pseudo code below.

Returns a boolean, takes in GameObject as a parameter

if this object is the same as the parameter

return false

otherwise

topthis = the top of this square

bottomthis = bottom of this square

leftthis = left of this square

rightthis = right of this square

topother = the top of the other object square

bottomother = bottom of the other object square

leftother = left of the other object square

rightother = right of the other object square

return not ( (bottomthis < topother) or  
 (topthis > bottomother) or  
 (leftthis > rightother) or  
 (rightthis < leftother))

* *draw*, which takes in a Graphics object and draws the GameObject of size 25 x 25 on the Graphics object with the passed in Color. The position is at the center. This method will then be called from the paintComponent’s draw.

**I suggest you test the above functionality in your program BEFORE moving on. Create some test cases of where at 25 x 25 square should collide with another 25 x 25 square.**

**In fact, almost none of the project (as describe in this document) is tested until you have completely finished the project. Don’t wait until the end to test your code in this or other parts. Create test cases and verify your code works BEFORE continuing on. I even tested my code to verify it worked!**

Player inherits from GameObject. It adds no member variables, but does add a host of new methods:

* *isOnGround*, which returns a boolean and takes in an ArrayList of ArrayList of GameObjects (the map data). It determines if the Player is directly above any of the GameObjects (i.e. is the player on the ground). (described later).
* *move*, which takes in two ints (how it should move relative to its current position) and an ArrayList of ArrayList of Game Objects (the map data); it returns a boolean, whether the move was successful. (described later).
* *collides*, which takes in a ArrayList of ArrayList of GameObjects (the map data), and determines if the player collides with any of the GameObjects nearby. It should call the GameObject collides method to determine it collides with any of the GameObjects in the ArrayList it needs to check. Don’t just check every object in the map to see if the player collides with it; although it would give you the correct true/false result the slowdown is significant for any large map and would result in reduced points.

Block inherits from GameObject. It adds nothing expect passes its own parameters into the super constructor

Victory Block is like Block, but with its own Color.

You may modify the art of the Player, Block, VictoryBlock, and GameObject as you see fit as long as the sizes stays the same and you can tell what they are.

**Reading the data file and the display**

(small version of file only to demonstrate the file format. A larger example that fills the screen is at the end)  
  
1 //the x position where the player starts, in blocks  
1 //the y position where the player starts, in blocks  
  
10 //the number of rows  
10 //the number of columns  
  
1 1 1 1 1 1 1 1 1 1 ///1 indicates a red block, 0 indicates nothing   
1 0 1 0 0 0 0 0 0 1 ///2 indicates the victory block, which is gray  
1 0 1 2 0 0 0 0 0 1  
1 0 1 1 1 0 0 0 0 1  
1 0 0 0 0 0 0 0 0 1  
1 0 0 0 0 0 0 0 0 1  
1 0 0 0 0 1 1 0 0 1  
1 0 0 0 0 1 0 0 0 1  
1 0 0 0 0 1 0 0 0 1  
1 1 1 1 1 1 1 1 1 1

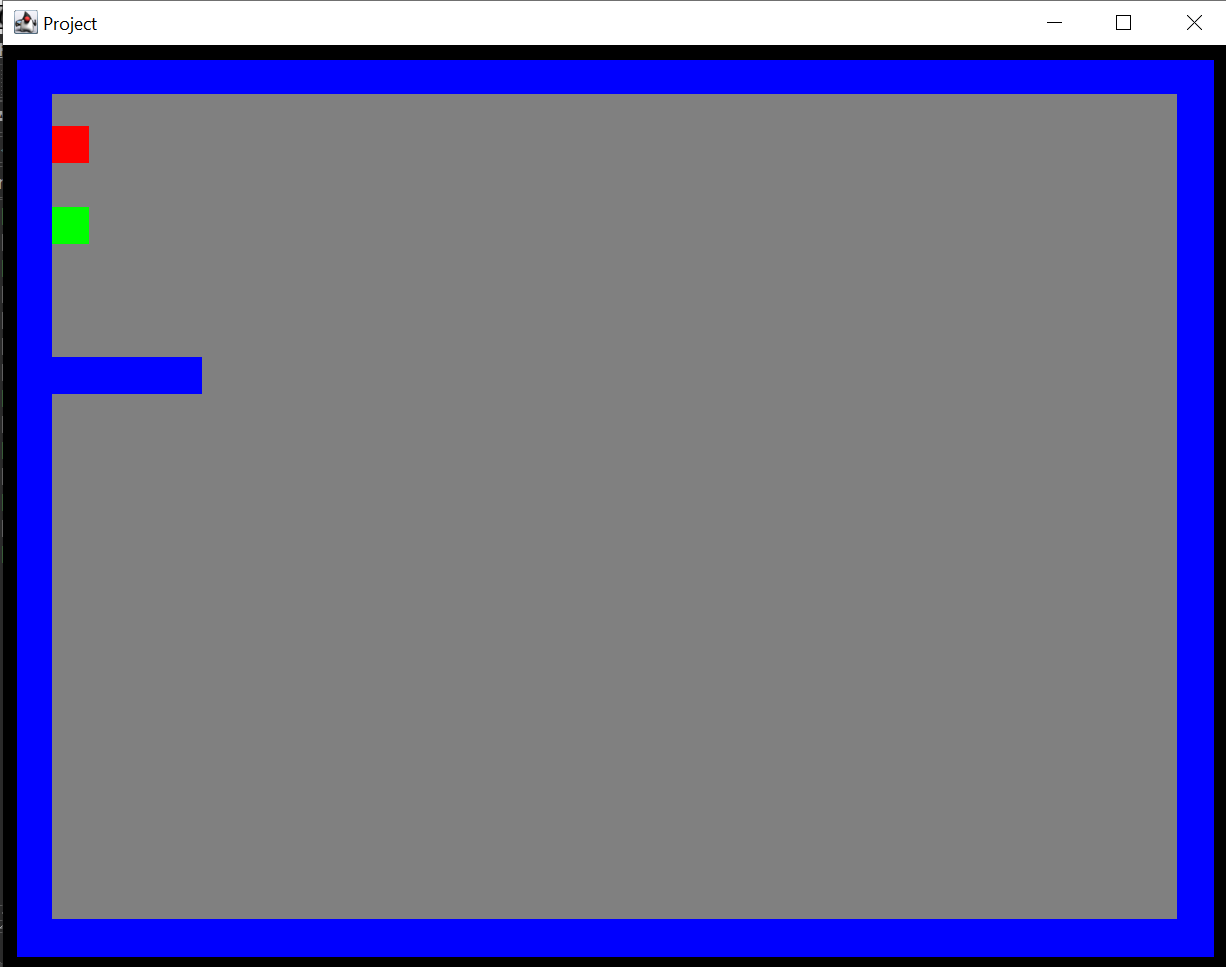
The above example file is the format for the game data file (with the exception that I added comments to help with readability). Notice it contains information about where the player starts and then the level data itself.

In your panel’s constructor

* You are to read in the data from a file. The actual file name you can hardcode, but your program should work with any level file. **A different text file will be provided to test your program with so be sure it works properly! (You can assume a correct file name)**
* Create blocks based on the data in the file.
* Create an ArrayList of ArrayList of GameObject, which will hold all the blocks you created based on the file.
* Initialize the player given the player’s information in the file.
* Note you must calculate the position of each block based on the size of the block and where it is in the 2d data list.
* I suggest that if there is no block, you set that place in the ArrayList of ArrayList to null; this way, you can check whether a block is at that spot or not (or create an extra “EmptyObject” class and set each of those positions to that object).
* In your panel’s paint component, you are to draw the GameObjects in the 2D ArrayList and the player and victory block by using a method we implemented earlier.

Longer file:

1   
1   
  
24   
32   
  
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1  
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1  
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1  
1 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1  
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1  
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1  
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1  
1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1  
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1  
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1  
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1  
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1  
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1  
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1  
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1  
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1  
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1  
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1  
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1  
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1  
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1  
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1  
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1



The above display matches the sample text file. **A different text file will be provided to test your program with, so be sure it works properly!**

**Functionality and Collision**

Implement isOnGround to do the following:

* It modifies the player’s position by 1 pixel lower than it currently is
* It checks to see if it collides with any of the blocks at its current position.
* It moves the player 1 pixel back to its original position.
* It returns whether it collided with anything while it was 1 pixel lower.

Implement move to do the following:

* Similar to isOnGround, move attempts to move by the x and y.
  + If it does not collide with anything, it keeps the player where it just moved it.
  + If it collided with anything, it moves back toe the previous point isOnGround.

Timer time!

The timer will do the following every 10 ms

* Gravity
  + When the player is not on the ground, apply N 1-pixel movements downward (note, you must actually do N single pixel moves and not one move of two pixels.) The amount of downward movements (N) starts at 1. After each 20 ticks of the timer (until N is 7), N increases by 1. Anytime the player is on the ground, N goes back to 1.
* Left and right movement!
  + While the player is holding the A key, move 1 pixel left.
    - The movement should be smoothed.
  + While the player is holding the D key, move 1 pixel right.
* Jump!
  + When the player is presses W and is currently on the ground, make the player “jump”. (gravity should also be reset if a jump occurs).
  + Similarly to gravity, you should have a jump variable set to 7 when the player presses W.
  + Move the player 1 pixel up jump number of times, and take off .1 of jump per timer tick (i.e. jump -= .1). If the player hit a ceiling, set jump to 0.
    - How do you think you can determine if the player collides with a ceiling block while the player is jumping? You may have to modify the classes in order to get this working.
* Determine if the player won by touching the victory block and use a JOptionPane to indicate victory. (Look up how to use it!). Then use System.exit(1) to quit.