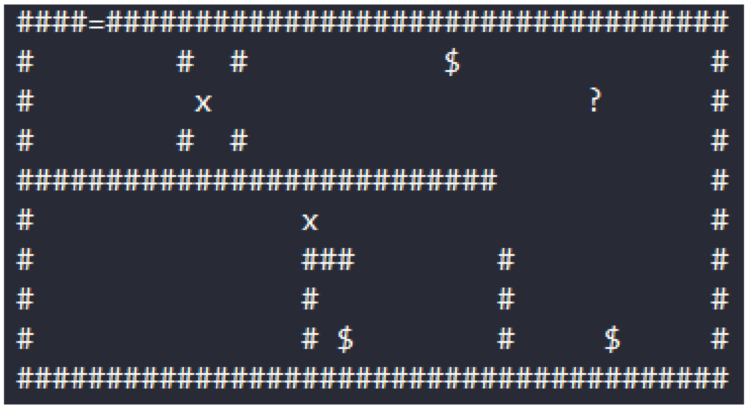
HOW TO MAKE A LEVEL

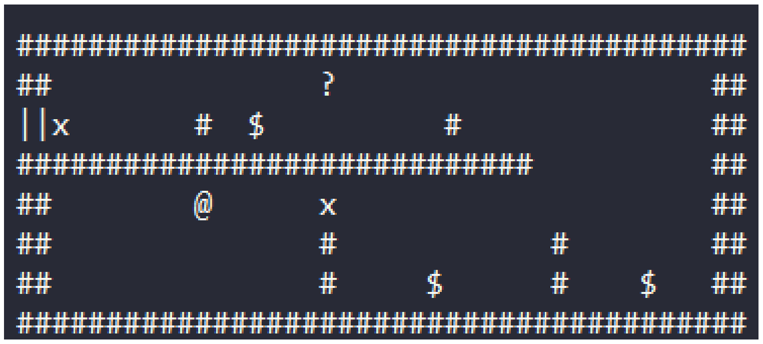
.MAP FILES

The .map file is the map files for the game. They are read by the LevelGenerator class and the characters representing the objects the player interacts with in the game. The current objects are Puzzle, Cipher, Prop, Trigger. Below are the **two** maps currently used for testing.

level1.map



```
####===================================
```



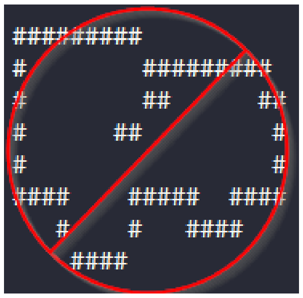
level2.map

GUIDE

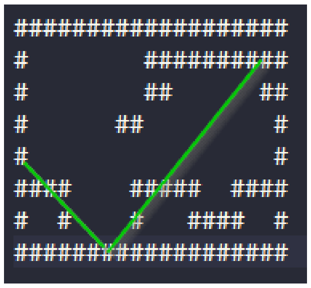
#	- **Wall**
|| or **=**	- **Door** (if using || door use ## walls on the east and west side of map to make it look even, see level2.map)
$	- **Prop**
?	- **Puzzle**
@	- **Cipher Puzzle** (appears as ? in Game)
x	- **Trigger** (a hidden object that prints text to screen when a player steps directly on it. Lower case x)
<(^.^)>	- **Player avatar** (not part of map but it is 7 characters long, remember this when designing levels)

MAP DIMENSIONS

Maps must be rectangular. If outer walls are not evenly aligned an array index error will occur.



```
#########
#        #########
#        ##      ##
#      ##         #
#                 #
####    #####  ####
   #    #   ####
    ####
```



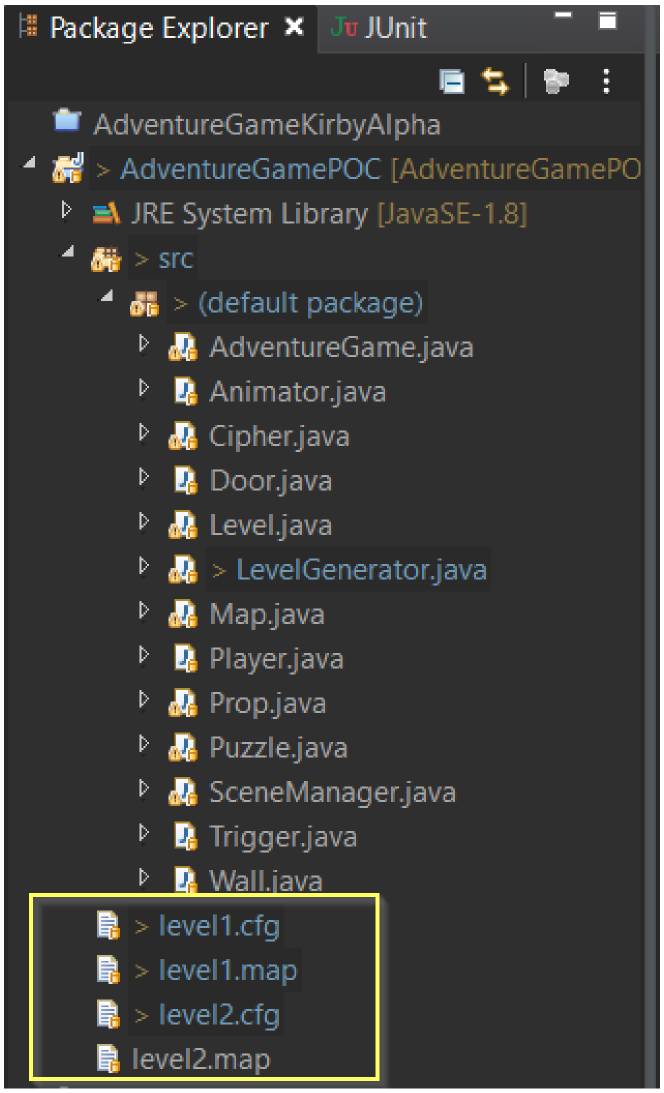
```
###################
#        ##########
#        ##      ##
#      ##         #
#                 #
####    #####  ####
#  #    #   #### #
###################
```

MAP FILE NAME

Maps are named based on their level number. So Level 1's map would be *'level1.map'*, Level 2 would be *'level2.map'*, and so on. **File names are case sensitive, all maps should be level#.map with a lower case l and # being the level number.**

MAP FILE LOCATION

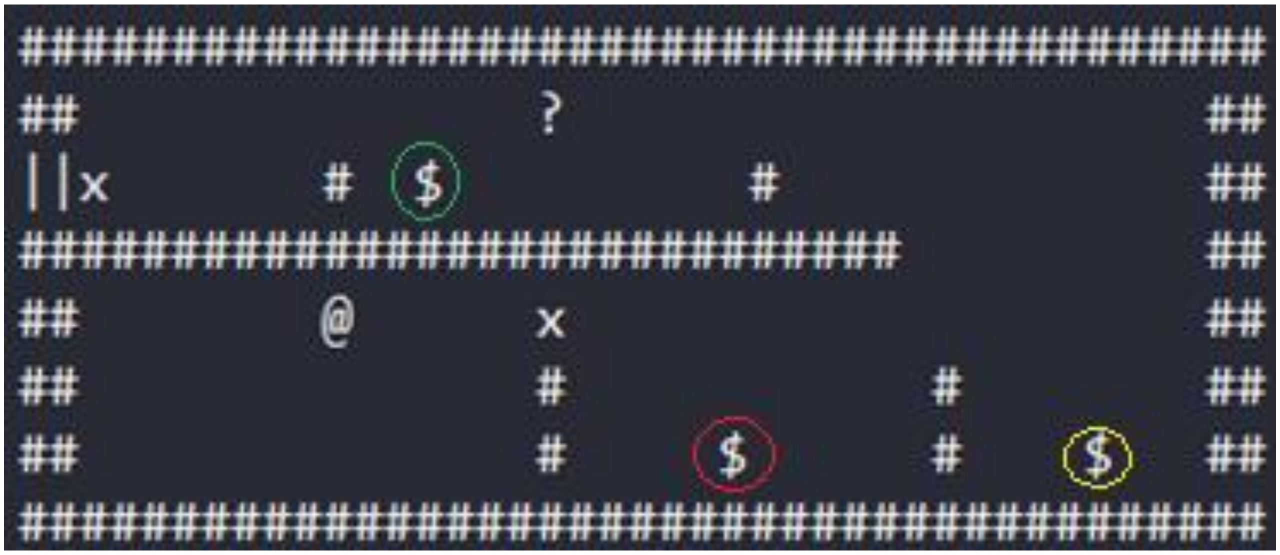
.map files need to be placed in the root of the project.



Map .map files in addition to the level configuration .cfg files must be placed in the root of the project folder. **If they are placed in the source folder (src), the application will have a runtime error.**

HOW LEVEL OBJECTS ARE GENERATED BY THE MAP

The map reads the objects based on their character starting at the 0,0 position on the map (the bottom left corner). They are then parsed row by row. The location of the object on the map (the x,y coordinate of object on the map is used when generating the objects for the Level.



```
#########################################
##               ?                     ##
||x       #  $          #              ##
#############################          ##
##        @      x                     ##
##               #            #        ##
##               #     $      #    $   ##
#########################################
```

For the above example of the Props generated from level2.map:

The Red circled prop is located at (23,1) and its information in the Level Configuration file is loaded first.

The Yellow circled prop is located at (35,1) and its information in the Level Configuration file is loaded second.

The Green circled prop is located at (13,5) and its information in the Level Configuration file is loaded third.

HAVE AN EQUAL NUMBER OF MATCHING OBJECTS ON MAP AS YOU HAVE IN THE CONFIG FILE

If you have 3 Puzzle, 2 Prop, 1 Trigger, and 1 Cipher, you need to make sure you have matching number of entries in the matching level config file (level#.cfg, # meaning the level number). The next part of the document demonstrates how to put Puzzles, Ciphers, Props, Triggers, and other constructor information in the Level config file.

.CFG FILES

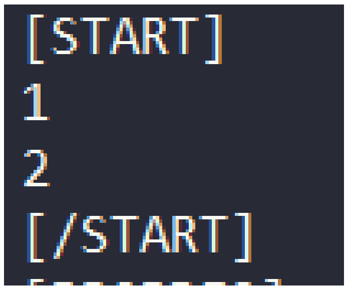
The .cfg file contains the strings other information to generate the Level. They are read by the LevelGenerator class and use a primitive XML-like system to gather the constructor information for each Levels game objects.

CONFIG FILE NAME

Config files are named based on their level number. So Level 1's config file would be *'level1.cfg'*, Level 2 would be *'level2.cfg'*, and so on.

Make sure the filename before the '.' matches the corresponding map. The files are case sensitive so remember lower case for all characters, or there will be a file I/0 exception.

PLAYER STARTING POSITION FOR LEVEL



```
[START]
1
2
[/START]
```

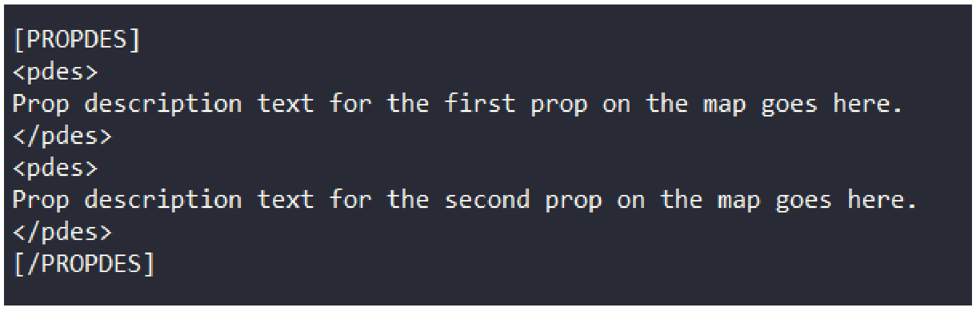
[START] is the delimiter for the LevelGenerator to start getting the players starting x and y position.

The first number on the line is the player's X starting position (so 1 in this example).

The second number on the line is the player's Y starting position (so 2 in this example).

[/START] is the delimiter to tell the LevelGenerator that it has finished gathering the players starting position.

PROP GENERATION (THE TEXT THE PLAYER READS WHEN THEY INTERACT WITH A PROP)



```
[PROPDES]
<pdes>
Prop description text for the first prop on the map goes here.
</pdes>
<pdes>
Prop description text for the second prop on the map goes here.
</pdes>
[/PROPDES]
```

[PROPDES]	The delimiter for the LevelGenerator to start getting the Props descriptive text for the level. In this example there are 2 Props in the level, so there are 2 prop descriptions, enclosed in <pdes> and </pdes>.
<pdes>	The delimiter for the LevelGenerator to grab the next line (or lines) of text until it reaches the </pdes> delimiter.
</pdes>	The delimiter to tell LevelGenerator that the current Prop description String it is reading is done.
[/PROPDES]	The delimiter to tell LevelGenerator that it has finished generating the levels Props.

PUZZLE GENERATION

Puzzle has 2 String variables that need to be generated, the Puzzle Question and the Puzzle Answer. In the example below we are generating 3 Puzzles in the level. Represented by **?** on map.

```
[PUZZLE]
<PQUESTIONS>
<pquestion>
Who wants to get an A+?
</pquestion>
<pquestion>
What year is it?
</pquestion>
<pquestion>
What season is it?
</pquestion>
</PQUESTIONS>
<PANSWERS>
<panswer>
We do!
</panswer>
<panswer>
2020
</panswer>
<panswer>
Winter
</panswer>
</PANSWERS>
[/PUZZLE]
```

[PUZZLE]	The delimiter used to tell the LevelGenerator to start generating Puzzles.
<PQUESTIONS>	The delimiter used to tell the LevelGenerator to start generating the Questions for the Level's Puzzles.
<pquestion>	The delimiter used to grab the next line (or lines) question and put it in a String.
</pquestion>	The delimiter used to tell the LevelGenerator that the current String is done.
</PQUESTIONS>	The delimiter used to tell the LeveLGenerator that it has reached the end of the Puzzle Question Strings.
<PANSWERS>	The delimiter used to tell LevelGenerator to start generating the Answers for the Level's Puzzles.
<panswer>	The delimiter used to tell LevelGenerator to grab the next line (or lines, for answers a single line or word would be best, but it can handle multiple lines).
</panswer>	The delimiter used to tell the LevelGenerator that the current String is done.
</PANSWERS>	The delimiter used to tell the LeveLGenerator that it has reached the end of the Puzzle Answer Strings.
[/PUZZLE]	The delimiter used to tell the LevelGenerator to end generating Puzzles.

TRIGGER GENERATION

The trigger is a hidden object that prints text to screen when a player steps directly on it. Lower case x on map. Two triggers are used in the example below.

```
[TRIGGER]
<trigger>
You hear the sound of footsteps behind you!
</trigger>
<trigger>
Ewwww what was that smell!
</trigger>
[/TRIGGER]
```

[TRIGGER]	Used to signify the start of Trigger Generation
<trigger>	Used to signify the start of the Trigger's String generation. Will gather line (or lines) of text until it finds </trigger>.
</trigger>	Used to signify the end of the current Trigger's String.
[/TRIGGER]	Delimited to signify the end of generating Triggers.

CIPHER GENERATION

Ciphers have 3 String variables that need to be generated, the Cipher Question, Cipher Answer, and the Cipher hint. Cipher is represented by **'@'** on the .map file, and uses the same '?' symbol as puzzles in the actual game. In the example below we are generating 1 Cipher in the level (can generate as many as you want).

```
[CIPHER]
<CQUESTIONS>
<cquestion>
You open a drawer of Scarlett's desk and find some financial documents….
It looks right before Scarlett died, she changed her will so her entire fortune
would go to someone she had just met 2 weeks prior – her new romantic partner
that she just began seeing. He was probably at the party too, but the only way
to find out is to translate the cipher.
That shows his name on this paper.
</cquestion>
</CQUESTIONS>
<CHINTS>
<chint>
The hint is 'EJTKU'. And the algorithm is X(alphabetic index)-2.
</chint>
</CHINTS>
<CANSWERS>
<canswer>
Chris
</canswer>
</CANSWERS>
[/CIPHER]
```

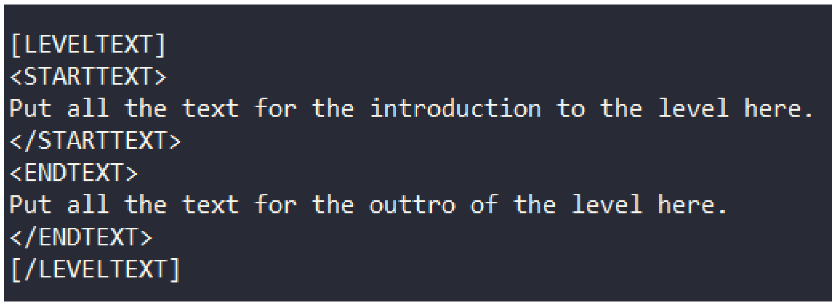
[CIPHER]	- Used to signify the start of Cipher Generation.
<CQUESTIONS>	- Delimiter used to tell LevelGenerator to start generating Cipher question Strings.
<cquestion>	- Delimiter used to tell LevelGenerator to start adding the next line (or lines) to a new Cipher Question String.
</cquestion>	- Delimiter used to tell LevelGenerator that the current Cipher Question String is complete.
</CQUESTIONS>	- Delimiter used to tell LevelGenerator that it is the end of the Cipher Questions for the Level.
<CHINTS>	- Delimiter used to tell LevelGenerator to start generating Cipher Hint Strings.
<chint>	- Delimiter used to tell LevelGenerator to start adding the next line (or lines) to a new Cipher Hint String.
</chint>	- Delimiter used to tell LevelGenerator that it has reached the end of the current Cipher Hint String.
</CHINTS>	- Delimiter used to tell LevelGenerator that it has reached the end of the Level's Cipher Hints.
<CANSWERS>	- Delimiter used to tell LevelGenerator that it has reached the start of the Cipher Answers.

CIPHER GENERATION CONTINUED…

<canswer>	- Delimiter used to tell LevelGenerator to start adding the next line (or lines) to a new Cipher Answer String.
</canswer>	- Delimiter used to tell LevelGenerator that it has reached the end of the current Cipher Answer String.
</CANSWERS>	- Delimiter used to tell LevelGenerator that it has reached the end of the Cipher Answers in the level.cfg file.
[/CIPHER]	- Used to signify the end of Cipher Generation.

LEVEL STORY TEXT GENERATION

Level’s print story text at the beginning and end of the level.



```
[LEVELTEXT]
<STARTTEXT>
Put all the text for the introduction to the level here.
</STARTTEXT>
<ENDTEXT>
Put all the text for the outtro of the level here.
</ENDTEXT>
[/LEVELTEXT]
```

[LEVELTEXT]	– signifies to LevelGenerator to start generating Level Story text.
<STARTTEXT>	- tells LevelGenerator to add the following line (or lines) to the level introduction text String.
</STARTTEXT>	- tells LevelGenerator that the Level intro text is complete.
<ENDTEXT>	- tells LevelGenerator to add the following line (or lines) to the level end text String.
</ENDTEXT>	- tells LevelGenerator that the Level ending text is complete.
[/LEVELTEXT]	- tells LevelGenerator that Level text generation is complete.

ORDER MATTERS FOR TEXT FOR EACH OBJECT

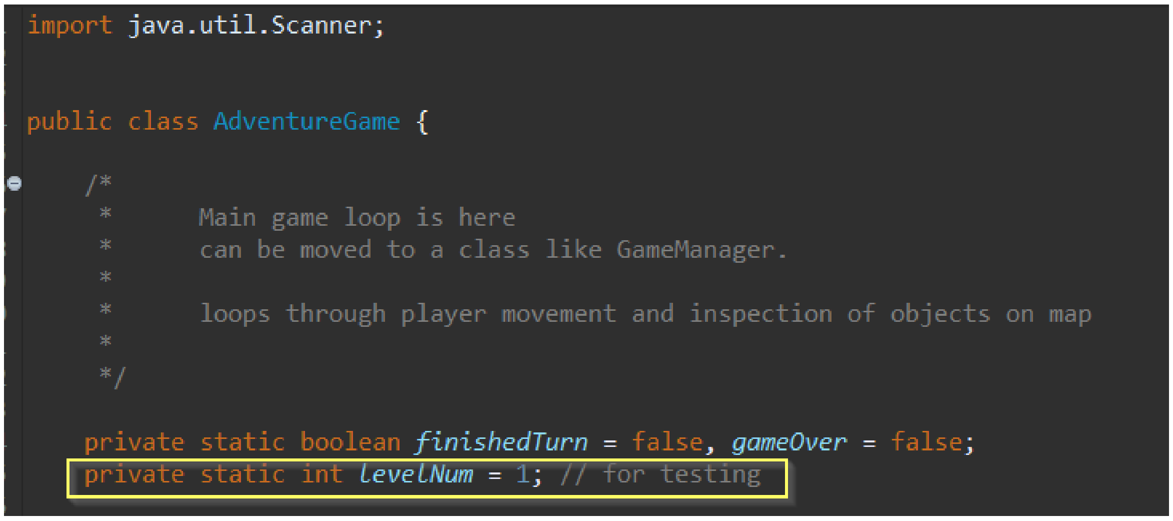
Make sure to put text in the level#.cfg file in the proper order of the object being generated. So the first Puzzle's Question and Answer should be the in the first blocks ie:

```
[PUZZLE]
<PQUESTIONS>
<pquestion>
First puzzle question.
</pquestion>
<pquestion>
Second puzzle question.
</pquestion>
</PQUESTIONS>
<PANSWERS>
<panswer>
First puzzle answer.
</panswer>
<panswer>
Second puzzle answer.
</panswer>
[/PUZZLE]
```

The order you put the block of Ciphers, Puzzles, etc does not matter. AS LONG AS THE SAME TYPE OBJECTS ARE GROUPED TOGETHER IN THE SAME FORMAT AS THE EXAMPLES ABOVE THINGS WILL WORK.

HOW TO TEST YOUR LEVEL

You can change the level1.map, level1.cfg, level2.map, and level2.cfg if you want to play around. If you want to make your own level number you need to import the .map and .cfg files to root of the project in Eclipse, and you can specific the level number to load in AdventureGame.java by changing the level number highlighted below.



```java
import java.util.Scanner;


public class AdventureGame {

    /*
     *      Main game loop is here
     *      can be moved to a class like GameManager.
     *
     *      loops through player movement and inspection of objects on map
     *
     */

    private static boolean finishedTurn = false, gameOver = false;
    private static int levelNum = 1; // for testing
```