Assignment 2 - Rogue Expanded

Welcome to Rogue Expanded

Welcome to Rogue Expanded, the advanced version of the base game you implemented in Assignment 1.

In this assignment, we will build on our existing game to add more features to make it a more fully-featured game.

Features we will add include:

- An inheritance structure for managing entities in the World (Player, Monster, Item classes)
- Reading in "game world" files to create game worlds from
- An improved game world
    - Managing an arbitrary numbers of entities in the world
    - Managing a map (Map class)
    - Consumable items in the world
- Saving and loading of player data files (new main menu commands: load, save)
- And other smaller improvements such as:
    - Player level-up
    - Player "damage perk" (bonus damage that applies only while in the world)
    - Minor main menu command changes (start command)

Note: All existing requirements from Assignment 1 still apply, except for any changes defined by the requirements for Assignment 2.

Additional non-feature requirements:

Since this system is now getting larger, it's important to consider designing your code to avoid redundant code through the effective use of **methods, loops, arrays (or ArrayLists)** and **class inheritance**.

You must also handle errors using **Exceptions** (creating your own custom exceptions classes, and throwing and catching them where appropriate).

Note: There will be marks associated with correct use of these features. Please refer to the marking scheme for more details, once released.

Academic Honesty

- All assessment items (assignments, test and exam) must be your own, individual, original work.
- Any code that is submitted for assessment will be automatically compared against other students' code and other code sources using sophisticated similarity checking software.
- Cases of potential copying or submitting code that is not your own may lead to a formal academic misconduct hearing.
- Potential penalties can include getting zero for the project, failing the subject, or even expulsion from the university in extreme cases.
- For further information, please see the university's Academic Honesty and Plagiarism website, or ask your lecturer.

Main Menu Commands

To accommodate the new features introduced in Rogue Expanded, we will need to make a change to the **start** command, to allow loading of worlds from game level files, as well as introduce two new commands, **load** and **save**, which will allow users to load and save their player data.

All other commands should stay working the same, as per the specifications from Assignment 1.

Starting a game (from a game world file)

The **start** command with no arguments behaves the same as in Assignment 1, but when the name of a game world file is provided as an argument to this command, rather than loading a default world, a new World object is created and initialized based on the contents of the file. A map of the area is loaded, and entities such as the players, monsters and items are also loaded into the world.

```
start <game world name>
```

For example:

```
 ____
|  _ \ ___   __ _ _   _  ___
| |_) / _ \ / _` | | | |/ _ \
|  _ < (_) | (_| | |_| |  __/
|_| \_\___/ \__, |\__,_|\___|
COMP90041   |___/ Assignment

Player: [None]  | Monster: [None]

Please enter a command to continue.
Type 'help' to learn how to get started.

> player
What is your character's name?
Bilbo
Player 'Bilbo' created.

(Press enter key to return to main menu)


 ____
|  _ \ ___   __ _ _   _  ___
| |_) / _ \ / _` | | | |/ _ \
|  _ < (_) | (_| | |_| |  __/
|_| \_\___/ \__, |\__,_|\___|
COMP90041   |___/ Assignment

Player: Bilbo 20/20  | Monster: [None]

Please enter a command to continue.
```

```
Type 'help' to learn how to get started.

> start example
...####...@.
.B.^##..o...
.........s.
~~~.s..+....
~~~~~~~...s.
~~~~~~~~~~~~

>
```

Note that in the example above, the game world name is called "**example**", and the command, therefore, loads a file called "**example.dat**" from the same directory as the program (refer to the section "Game World File Structure" for more details on the contents of the file).

The command will fail if no player has been created or **load**ed (see the **load** command below), but will not fail if no monster was created, since monster data is available from within the game world file.

For example:

```
 ____
|  _ \ ___   __ _ _   _  ___
| |_) / _ \ / _` | | | |/ _ \
|  _ < (_) | (_| | |_| |  __/
|_| \_\___/ \__, |\__,_|\___|
COMP90041   |___/ Assignment

Player: [None]  | Monster: [None]

Please enter a command to continue.
Type 'help' to learn how to get started.

> start
No player found, please create a player with 'player' first.

(Press enter key to return to main menu)
```

Save / Load (player data)

Two new commands **save** and **load** should now also be made available from the main menu. These will allow users to either load existing player data or save player data for use later (refer to the section "Saving and Loading Player Data" for the details of how these commands work).

Improved Game World

Rogue Expanded features an improved world with much more to do and explore.

Each world is an area with terrain that players and monsters can move over, as well as terrain that players cannot be moved across. Players are also, as before, constrained by the bounds of the world.

The world is populated with the player, monsters, and useful items which can be picked up by the player and which have different effects, such as healing the player, increasing the player's damage, and levelling up the player.

The goal of each world is to reach the **warp stone** ("@") at the end of each level. Reaching the warp stone will reward the player with an increase in level by 1, and then transport the player back to the main menu to begin a new adventure.

The world from the world file "example.dat":

```
...####...@.
.B.^##..o...
..........s.
~~~.s..+....
~~~~~~~...s.
~~~~~~~~~~~~
```

From the above map:

- "." is normal ground, and indicates a traversable space that monsters and players can move over.
- "#" is mountainous terrain, and indicates a space that cannot be traversed over.
- "~" is an area of water, and indicates a space that cannot be traversed over.
- "B" is the player Bilbo.
- "s" and "o" are monsters on the map.
- "^", "+", and "@" are items on the map.

The player can still move around the world and encounter enemies to battle, as well as now being able to pick up items.

Map and Entities in the World

In our improved game world, our **World** has a specific layout. This includes the look of the terrain, and whether or not the terrain can be traversed or not.

In our improved game world, we have a concept of "entities", which are "things" that exist somewhere in the world.

Entities that will exist in the World include:

- **Player**
- **Monster**
- **Item**

You should create an **Entity** class for grouping common functionality required by all entities in the world. You can use any inheritance structure you think appropriate, but at a minimum, an Entity class must be used as a parent or ancestor for those classes.

Consider using extra classes / abstract classes / interfaces to help you generalize and reuse code. For example, Players and Monsters behave in some similar ways, perhaps a common super class would be useful here.

Note that our World should be able to handle an arbitrary number of entities.

Details of these classes will be outlined in the next section "Class Description and Responsibilities".

Guidance: Class Description and Responsibilities

To give some direction in the design of your system, here is an outline of the **required**, and **suggested but optional** classes for your system. Only the required classes must be used by your system, with the suggested classes highly recommended (but not mandatory). As long your program implements the **required** classes, the exact inheritance hierarchy and use of extra classes is up to you.

World class (Required)

The overall class for each world. Holds map data and entities of the world. It would make sense for this class to manages interactions between entities (such as triggering battles, or detecting the pickup of items by the player). Manages the overall rendering of the world, perhaps delegating to some rendering to other classes.

Map class (Optional)

There is a lot of functionality related with managing the "map" for the world (the look of the terrain, and whether or not it is traversable). It would make sense to group this functionality out into a separate class. The map can manage the rendering of the terrain, and allow functionality for checking whether a particular location on the map is traversable or not. It would also make sense for the map to hold the dimensions of the map / world.

Entity class (Required)

The world manages a lot entities within it. It would make sense for entities to know how to render themselves, and hold positional information about where they are in the world.

Unit class (Optional)

Players and Monsters are similar in many ways. It would make sense to hold common information and functionality here. What common functionalities can you think of?

Player class (Required)

The player the user controls. Any data and functionality specific to what the player should hold or should be able to do should go in here. Consider what different entities the player should be able to interact with, and use that as guidance to what methods should appear on the player.

Monster class (Required)

Monsters in the world. Data and functionality specific to monsters should go in here. Consider what

actions monsters should be able to do, and what actions should be allowed to be done on them.

Item class (Required)

Items in the world. Consider how these should be used or interacted with.

Guidance: Game World Loop

Since we have to manage more things in our World, the game loop now needs to do much more than before. An outline of the things that the game loop for our world should do is outlined below, **including the order in which those things should be done.**

For each loop in the game loop:

- World and entities in the world are rendered, including the map terrain.
- Await user command.
- If the user entered the "home" command, return to the main menu.
- Monsters make their movement (not updated on screen immediately, will be updated on next render).
- The Player makes their movement (not updated on screen immediately, will be updated on next render).
- If a player and monster are on the same cell, a battle occurs.
    - If player loses, return to main screen
    - If player wins, defeated monster(s) are removed from the world.
- If a player and item are on the same cell, the item is picked up and removed from the world.

Note that if an unrecognized command or an **empty command** is entered by the user, the game loop should simply continue into its next iteration. The implication of this is that monsters close to the player will still move toward the player, and trigger battles if they move into the same cell as the player.

Note also that where we need to step through a collection of entities one by one, the order of iteration should be based on the order in which the entities were added into the world, which for worlds loaded from a game world file means the order at which they appeared in the file (refer to "Game World File Structure" for the details).

Monster Movement & Battle

Monsters can now move around in the world toward the player, if the player is close enough to it.

When the player is within **2 cells** of the monster (i.e., anywhere in a **5x5 area** around the monster's location), the monster will try to move toward the player to attack.

For example, a player that occupies any of the "." locations below would cause the monster (m) to move toward the player.

```
.....
.....
..m..
.....
.....
```

An example of a player moving toward a monster, and the monster detecting a player and moving toward the player:

```
...####...@.
.P.^##..o...
.........s.
~~~.s..+....
~~~~~~~...s.
~~~~~~~~~~~

> s
...####...@.
...^##..o...
.P........s.
~~~.s..+....
~~~~~~~...s.
~~~~~~~~~~~

> d
...####...@.
...^##..o...
..P.......s.
~~~.s..+....
~~~~~~~...s.
~~~~~~~~~~~

> d
...####...@.
...^##..o...
...P......s.
~~~s...+....
~~~~~~~...s.
~~~~~~~~~~~
```

Players being chased by a monster can move away from the monster to avoid battling the monster:

```
> a
...####...@.
...^##..o...
..Ps......s.
~~~....+....
~~~~~~~...s.
~~~~~~~~~~~

> a
...####...@.
...^##..o...
.Ps.......s.
~~~....+....
~~~~~~~...s.
~~~~~~~~~~~

> a
...####...@.
...^##..o...
Ps........s.
~~~....+....
~~~~~~~...s.
~~~~~~~~~~~
```

Note also that it is possible for players to "dodge past" a monster (since rogues are agile!), by moving into a cell that a monster occupies just as a monster is moving out of the cell toward the player, as a way of avoiding the monster.

```
> a
...####...@.
...^##..o...
Ps........s.
~~~....+....
~~~~~~~...s.
~~~~~~~~~~~

> d
...####...@.
...^##..o...
sP........s.
~~~....+....
~~~~~~~...s.
~~~~~~~~~~~

> d
...####...@.
...^##..o...
.sP.......s.
~~~....+....
~~~~~~~...s.
```

```
~~~~~~~~~~~~

> d
...####...@.
...^##..o...
..sP......s.
~~~....+....
~~~~~~~...s.
~~~~~~~~~~~~
```

Monsters can only move North, South, East or West, similar to players.

Monsters will always prefer moving toward the player in the East or West direction first, but if the target cell is blocked in those directions they will try moving in the North or South direction instead, if it brings the monster closer to the player. If the monster is still blocked from movement, the monster instead does not change position.

> Note: If the player enters an unrecognized command or the **empty command** which causes the player to stand still, monsters close by can still move toward the player and initiate battle.

Battle

If a monster ends up in the same cell as a player, a battle is initiated, as per Assignment 1.

If more than one monster is in the same cell as the player, then a battle is initiated per monster, one after the other.

Whenever a monster loses a battle (i.e., reaches 0 health or less), it should be removed from the world permanently.

After a battle completes, with the player winning, control is returned to the game world loop.

```
...####...@.
...^##..o...
..P.......s.
~~~s...+....
~~~~~~~...s.
~~~~~~~~~~~~

> d
Poring encountered a Slime!

Poring 20/20 | Slime 5/5
Poring attacks Slime for 2 damage.
Slime attacks Poring for 1 damage.

Poring 19/20 | Slime 3/5
Poring attacks Slime for 2 damage.
Slime attacks Poring for 1 damage.

Poring 18/20 | Slime 1/5
```

```
Poring attacks Slime for 2 damage.
Poring wins!

...####...@.
...^##..o...
...P......s.
~~~....+....
~~~~~~~...s.
~~~~~~~~~~~~

>
```

If the battle ended with the monster winning instead, then the player loses and is taken back to the main menu.

```
...####...@.
...^##.o....
.......B..s.
~~~....+....
~~~~~~~...s.
~~~~~~~~~~~~

>
Bilbo encountered a Orc!

Bilbo 18/20 | Orc 15/15
Bilbo attacks Orc for 2 damage.
Orc attacks Bilbo for 4 damage.

Bilbo 14/20 | Orc 13/15
Bilbo attacks Orc for 2 damage.
Orc attacks Bilbo for 4 damage.

Bilbo 10/20 | Orc 11/15
Bilbo attacks Orc for 2 damage.
Orc attacks Bilbo for 4 damage.

Bilbo 6/20 | Orc 9/15
Bilbo attacks Orc for 2 damage.
Orc attacks Bilbo for 4 damage.

Bilbo 2/20 | Orc 7/15
Bilbo attacks Orc for 2 damage.
Orc attacks Bilbo for 4 damage.
Orc wins!

(Press enter key to return to main menu)


 ____
|  _ \ ___   __ _ _   _  ___
| |_) / _ \ / _` | | | |/ _ \
|  _ < (_) | (_| | |_| |  __/
|_| \_\___/ \__, |\__,_|\___|
```

```
COMP90041   |___/ Assignment

Player: Bilbo -2/20  | Monster: [None]

Please enter a command to continue.
Type 'help' to learn how to get started.

>
```

Items & Item Use

Items

When a player moves into a cell in which an item is located, the item is picked up by the player, and the item is removed from the world.

The items that can be found in the world, and their effects when picked up, are as follows:

- + healing item - instantly heals the player to full health.
- ^ damage perk - increases the player's attack damage by 1, per damage perk collected. The damage bonus granted by this perk is reset when the player leaves the world (via the home command or on death).
- @ warp stone - the goal of each level is to collect this item. Collect this to **level up the player** by 1 level and exit the world (return back to the main menu).

Only Players can pick up items. Monsters can move into tiles with items on them. The entity shown on the map will be the entity added to the world the earliest.

If there is both a monster and an item on the same cell as the player, the battle occurs before the item is picked up.

Item Use

When an item is picked up, the following messages are shown for the respective items:

Healing item **+**: "Healed!"

```
...####...@.
....##..o...
.........s.
~~~...P+....
~~~~~~~...s.
~~~~~~~~~~~

> d
Healed!
...####...@.
....##.o....
.........s.
~~~....P....
~~~~~~~...s.
~~~~~~~~~~~

>
```

Damage perk **^:** "Attack up!"

```
...####...@.
..P^##..o...
.........s.
~~~.s..+....
~~~~~~~...s.
~~~~~~~~~~~~

> d
Attack up!
...####...@.
...P##..o...
.........s.
~~~s...+....
~~~~~~~...s.
~~~~~~~~~~~~

>
```

Warp stone **@**: "World complete! (You leveled up!)"

```
> d
...####..P@.
....##......
.........s..
~~~.........
~~~~~~~...s.
~~~~~~~~~~~~

> d
World complete! (You leveled up!)

(Press enter key to return to main menu)
```

Note that the levels gained by the player persist when the player leaves the world. This means that the more worlds the player completes, the higher the level, and the more powerful, the player becomes.

Game World File Structure

Your main menu **start** command in Rogue Expanded should allow the user to load game world files to generate the world.

The structure of the game world files is as follows:

```
<map width> <map height>
<lines of map data>
player <x> <y>
monster <x> <y> <monsterName> <health> <attack>
item <x> <y> <symbol>
```

Example game world file "simple.dat":

```
12 6
...####.....
....##......
............
~~~.........
~~~~~~~.....
~~~~~~~~~~~~
player 1 1
monster 4 3 Slime 5 1
monster 10 2 Slime 5 1
monster 10 4 Slime 5 1
monster 8 1 Orc 15 4
item 7 3 +
item 3 1 ^
item 10 0 @
```

Map Data

In the above game world file, the first line indicates the map will be 12 cells wide and 6 cells high.

The following 6 lines then contain the terrain data for the map. "#" and "~" indicate non-traversable tiles, and any other symbol indicates a traversable tile. The map remembers the look of the terrain (i.e., the symbol) for each cell.

Entity Data

Following that, is the entity data: the player data first, followed by the monster data, and then the item data.

Each of these lines should be parsed, and the appropriate objects created for the entities and added to the world. Note that for the player, a new player object should not be created, but rather the player should be set to the location on the world specified by the line.

Entities should be added into the world in the order of the lines for the entities (player, monster, item). If two entities occupy the same location in the world, the entity with to be rendered first (and then overwritten), is the entity that was added into the world earlier.

The position of the entities provided in the file will be unique, i.e., no two entities will have the same starting position on the map.

Handling Errors

Your program should check for some basic errors that can occur when trying to read in the map file.

If a map could not be loaded because a file with that name does not exist, a custom **GameLevelNotFoundException** should be thrown rather than the default **FileNotFoundException** (you should catch the FileNotFoundException when trying to load a file that does not exist, and throw a GameLevelNotFoundException instead). You should implement this exception as a custom exception implemented by you.

Your program must also catch this exception in a suitable method and then display the message indicating that the map file could not be loaded: "Map not found."

General **IOExceptions** should be handled by printing the message "An error occurred while loading the file.", if any occur.

You must handle errors working with the map file using exceptions and try-catch blocks.

Saving and Loading Player Data

Two new commands **save** and **load** should now also be made available from the main menu. Their behaviour is described as below.

Save

The **save** command should allow the user to save their player data.

If a player has already been created via the **player** command, then the **save** command will save the player's name and level into a file named **player.dat**. If the file already exists, the data in the file will be overwritten. If the file does not exists, then it will be created.

```
 ____
|  _ \ ___   __ _ _   _  ___
| |_) / _ \ / _` | | | |/ _ \
|  _ < (_) | (_| | |_| |  __/
|_| \_\___/ \__, |\__,_|\___|
COMP90041   |___/ Assignment

Player: Bilbo 20/20  | Monster: [None]

Please enter a command to continue.
Type 'help' to learn how to get started.

> save
Player data saved.

>
```

The structure of the save file should be the name of the player, and the player's level. For example, saving a player called "Bilbo" with a level of 1 will write the following the following to the **player.dat** file:

```
Bilbo 1
```

If no player data is available to be saved (because a player has not yet been created or loaded), then the command will fail with an error message.

```
 ____
|  _ \ ___   __ _ _   _  ___
| |_) / _ \ / _` | | | |/ _ \
|  _ < (_) | (_| | |_| |  __/
|_| \_\___/ \__, |\__,_|\___|
COMP90041   |___/ Assignment

Player: [None]  | Monster: [None]
```

```
Please enter a command to continue.
Type 'help' to learn how to get started.

> save
No player data to save.

>
```

Load

The **load** command should allow the user to load previously saved player data into the game. If a player has already been created via the **player** command or via a previous **load**, that player is replaced with the player from the save file. Players loaded from the saved file should be healed to full health.

```
 ____
|  _ \ ___   __ _ _   _  ___
| |_) / _ \ / _` | | | |/ _ \
|  _ < (_) | (_| | |_| |  __/
|_| \_\___/ \__, |\__,_|\___|
COMP90041   |___/ Assignment

Player: [None]  | Monster: [None]

Please enter a command to continue.
Type 'help' to learn how to get started.

> load
Player data loaded.

> player
Bilbo (Lv. 1)
Damage: 2
Health: 20/20

(Press enter key to return to main menu)
```

If no previously saved data exists (because the **player.dat** file could not be found), then the command fails with an error message.

```
 ____
|  _ \ ___   __ _ _   _  ___
| |_) / _ \ / _` | | | |/ _ \
|  _ < (_) | (_| | |_| |  __/
|_| \_\___/ \__, |\__,_|\___|
COMP90041   |___/ Assignment

Player: [None]  | Monster: [None]

Please enter a command to continue.
```

```
Type 'help' to learn how to get started.

> load
No player data found.

>
```

Rogue Expanded

Getting Started

Scaffold & Your Assignment 1 Code

Some scaffolding code has been created for you for this project (including empty files for the classes from Assignment 1: GameEngine, World, Player, Monster).

Copy the code you have from Assignment 1 into this scaffolding code to get started.

You will need to refactor your existing code to take advantage of the new features you have learned over the last few weeks (including inheritance and arrays).

Test Harness

We also provide for you an extra class for you to use for testing purposes called "TestHarness". We will not look at this class or test it at all.

The TestHarness class is essentially a (currently empty) driver program for you to be able to quickly create objects with, and to call methods on them to test them. For example, you may wish to simply create a World object and call methods on that directly to test some of your features, without having to create a new player using the main menu system each time.

You can run the test harness using the following command:

```
javac *.java
java TestHarness
```

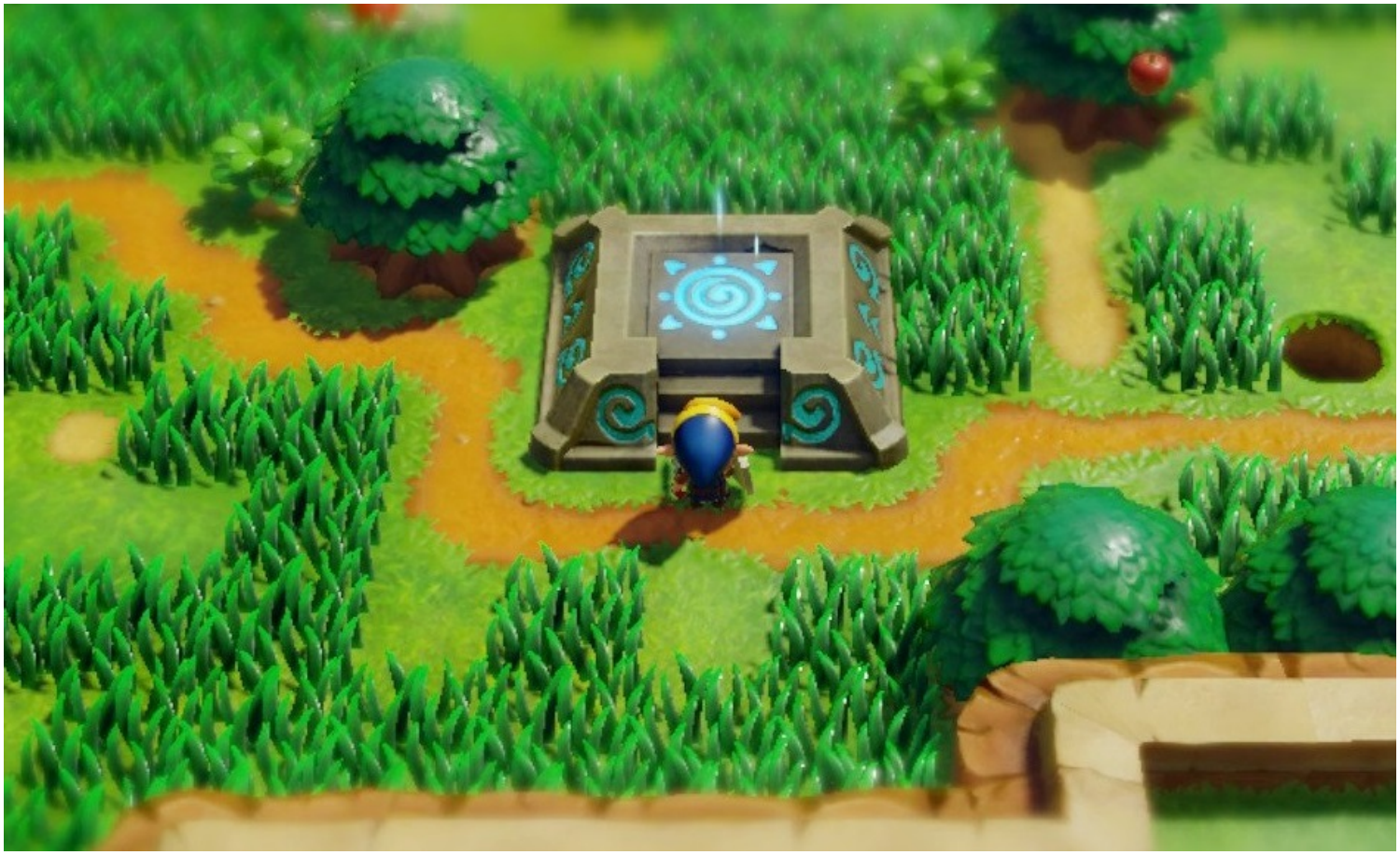
Game World Files

A game world file "simple.dat" has also been supplied with the scaffold code. You may use it to test your program. More game world files may be made available later. Feel free to also make and share your own game world files on the discussion board.

Test Cases

Some test cases will be made available to you later to help guide your coding (here, on edstem). Note that while these will help you check some parts of your system, it is still important to test your program in full yourself as well.

Bangarang!!!



Marking Scheme

Warning! Please make sure your code runs on edstem. If your code does not compile on edstem, 0 marks will be given.

COMP90041 Assignment 2: Marking Scheme

Student: ***[Student ID goes here]***

Program Presentation

Including: layout and style, readability, adherence to coding expectations and conventions, general care and appearance. The full marks for this section of marking is **6**.

Deductions

Some subset of the following lines will be selected by the marker.

Gain **0.5** marks for any of the achievements listed below.

- all choices of variable names were meaningful;
- all choices of method names were meaningful;
- variable and method names follow Java convention (camelCase);
- constant names follow Java convention (UPPER_SNAKE_CASE);
- class names follow Java convention (UpperCamelCase);
- comments were sufficient and meaningful;
- consistent bracket placement;
- indentation was consistent;
- whitespace was used to separate logical blocks of code;
- authorship statement (name, university email, student number) provided;
- all magic numbers (essentially numbers other than 0 or 1) were assigned to constants;
- overall care and presentation, such as particularly good comments, up to **+1.0** mark;

Deductions

- stylistic issue, if major **-1.0** mark per issue; if minor, **-0.5** mark per issue

Total marks for this section: **/ 6**

Other comments from marker: ***[Comments go here]***

Structure and Approach

Including: decomposition into methods, declaration of instance variables at the appropriate locations, choice of parameters to methods.

The full marks for this section of marking is **13**.

Deductions

Some subset of the following lines will be selected by the marker.

Gain **1** mark for any achievement listed below.

- Good use of methods;
- No methods were too long or too complex;
- code was not duplicated (tasks to be done in more than one place are in method);
- simple algorithmic approach to solving the problem;
- clearly structured code;
- no more than 3 `static` methods (including main) were used -- most should be bound to objects;
- no more than 4 `static` variables were used -- most should be non-static ("`final static`" constants are not variables);
- appropriate use of encapsulation; limiting scope of variables to `private` except where reasonable justification is given.
- constants were defined as `final static`;
- exceptions were appropriately thrown when required, and try-catch blocks were appropriately used (this usually means the catch is in a different method from the throw, though no always);
- custom GameLevelNotFoundException was implemented with appropriate constructors;
- classes were implemented using inheritance relationships where appropriate;
- all required classes (GameEngine, World, Player, Monster) were used in the implementation of the system.

Deductions

- structural issues, if major **-2.0** marks per issue, otherwise **-1.0** per issue.

Total marks for this section: **/ 13**

Other comments from marker: ***[Comments go here]***

Program Execution

Including: compilation, execution on test data, output presentation and readability.

Programs that do not compile in the test environment will lose all marks in this section. Be sure to verify your submission and **check the output** before you say "finished" to yourself.

The full marks for this section of marking is **21**.

Marks awarded

Gain **0.75** marks per test case passed.

Deductions

Gain **0.5** marks for tests with ***slightly*** different output (e.g., small changes in whitespace; if the marker says the difference is not slight, that is final. This should not occur if the available test cases are all checked.);

Gain **0** for other failed tests

Visible tests passed: / 9

Assessment tests passed: / 19

Total tests passed: / 28

Total marks for this section: **/ 21**

Total Marks for the Assignment: / 40

Overall comments from marker: ***[Comments go here]***

Assignment Marker: ***[Assignment marker name goes here]***

If you have any questions regarding your mark, please contact the lecturers

Assessment

This project is worth 15% of the total marks for the subject. Your Java program will be assessed based on the correctness of the output as well as the quality of code implementation. A detailed marking scheme will be released here on edstem.

Automatic tests will be conducted on your program by compiling, running, and comparing your outputs for several test cases with generated expected outputs. The automatic test will deem your output wrong if your output does not match the expected output, even if the difference is just having extra space or missing a colon. Therefore, it is crucial that your output follows exactly the same format shown in the provided examples.

> Passing the tests we provide here on edstem does not mean you will score full marks. Each submission will further be subject to hidden tests as well as manual inspection by our teaching team. However, if you do not pass these tests, it guarantees you will lose marks. Click "mark" often.

The syntax **import** is available for you to use standard java packages. However, DO NOT use the package syntax to customize your source files. The automatic test system may not deal with customized packages. If you are using Netbeans, IntelliJ or Eclipse for your development, be aware that the project name may automatically be used as the package name. You must remove lines like:

```
package Assignment1;
```

at the beginning of the source files when you copy them to edstem. Otherwise, the automatic tests may fail and tell you so.

Submission

Starter code has been provided to you here on the edstem platform, but you are welcome to develop the project outside of the edstem environment, using your own IDEs.

Submission is made via edstem, however, and will be whatever code is saved on the Code Challenge when the project deadline closes.

⚠ Your code MUST compile and run here on edstem. If your code does not compile we cannot mark it and you risk getting 0 marks.

"I can't get my code to work on edstem but it worked on my local machine" is not an acceptable excuse for late submissions. In addition, **submissions via email or Canvas will not be accepted.**

Late submissions will not be accepted, except for extensions granted on medical grounds.

Be sure to copy your code into your Code Challenge workspace well before the deadline to avoid being locked out of submissions last minute.

Only the last version of your code will be graded. It is highly recommended that you update your code here on edstem frequently and well before the submission deadline. Last-minute "connection errors" are not a valid excuse.

ℹ Click on the "mark" button regularly. It will test your code under the conditions that will be used for assessment. It is possible for code to *seem* to work when used with the keyboard and screen, but not work in the assessment environment.