INFO1110 / COMP9001 Assignment 2

Goosechasers: A Titled Goose Game

Milestone Deadline: 11:59 PM Monday, 28th October 2019 AEST (Week 12)

Final Deadline: 11:59 PM Monday, 4th November 2019 AEST (Week 13)

Weighting: 10% of the final assessment mark.

> *You are a goose. Beautiful. Majestic. Unstoppable. In the wake of your majesty, even the most intense of human passions are but fleeting whispers in comparison - for not only are you a goose, you are **the** Goose: Her Excellency, Duchess Hilda von Goosensmirch the Red, Countess of Loch Lanore and Grandmaster of the Order of the Silver Leek. There are those, however, that claim to see you for what you truly are: a terror to society; an agent of chaos - an avatar of destruction.*
>
> *The Goosechasers, as they call themselves, belong to a nefarious order of humans (and human-adjacent creatures) dedicated to your capture and suppression. Thanks to an unfortunate series of events, you, Duchess Hilda von Goosensmirch the Red, Countess of Loch Lanore and Grandmaster of the Order of the Silver Leek, have been cornered by one or more of these Goosechasers - a scenario most sub-optimal for you and yours.*
>
> *You are, of course, a goose, and in all your wisdom and majesty, are privy to many a means of response to this situation. Perhaps you are feeling merciful, and simply wish to evade the Goosechasers until they are no longer a problem - or perhaps you have suffered their existence long enough, and must remind them that they have no right to encroach upon your domain.*

Picture Credits: Pixabay, Uploaded by Alexas_Fotos on June 20th, 2018.

Overview

Description

For this assignment, you will write a game that simulates a *wild goose chase*. The player will control a goose - a *creature* - that moves from one location to another, while avoiding other *creatures* that wish to harm it. The goose may attempt to frighten other creatures into running away (permanently) by *honking* at them, though some creatures will be more difficult to frighten than others. If the goose is caught by another creature that it cannot frighten, the player loses. If the goose successfully navigates to a location that allows it to *flee* from the other creatures, or successfully frightens away all other creatures on the map, the player wins.

Implementation details

Your program will be written in Python 3. The only Python module that you are allowed to import is `sys`. To help you begin, a scaffold has been provided. This scaffold consists of the following files:

- `game.py`, which should contain the main body of your program.
- `preprocessing.py`
- `creature.py`
- `location.py`
- `item.py`
- `creature_test.py` (only relevant to final submission; see **page 29** for details)

Your entire program must be contained in these files. It is recommended that you implement the functions in these files to the best of your ability, and use them to help you complete the assignment. You may create new functions as you see fit.

Additionally, a few sample configuration files have been given to help you start up your program.

Help and feedback

You are encouraged to ask questions about the assignment on the discussion board, on Ed.

Please ensure your code is comprehensible before requesting feedback. We recommend that your code adheres to the PEP 8 style guide, and is commented appropriately.

Staff may make announcements on Ed regarding any updates or clarifications to the assignment. You can ask questions on Ed using the assignments category. Please read this assignment description carefully before asking questions. Please ensure that your work is your own and you do not share any code or solutions with other students.

Submission

You will submit your code on two separate assignment pages on **Edstem** - one for your *Milestone submission* and one for your *Final submission*. You are encouraged to submit multiple times. After each submission, the marking system will automatically check your code against public test cases.

These public tests do not cover all parts of the specification and your code. The complete test suite contains both public and hidden test cases, and your code will not be run through this suite until after the assignment deadline.

Please ensure you carefully follow the assignment specification. Your program output must exactly match the output shown in the examples.

Warning: Any attempts to deceive or disrupt the marking system will result in an immediate zero for the entire assignment. Negative marks can be assigned if you do not properly follow the assignment specifications, or your code is unnecessarily or deliberately obfuscated.

Milestone

To ensure you are making regular progress on this assignment, you must have achieved a minimum level of functionality in your submission by **October 28th, 11:59 PM AEST** (Week 12 Monday) to receive a portion of the marks. See the *Milestone Submission* section at the end of this document for further details.

Configuration Files

The program will be given 4 extra command line arguments when it is run:

```
$ python3 game.py <PATHS> <ITEMS> <CHASERS> <EXITS>
```

These arguments (let's call them **configuration files**) will be used to generate `Location`, `Item`, and `Creature` objects that you will need to run your program. The `Location`, `Item`, and `Creature` classes are to be defined in the `location.py`, `item.py`, and `creature.py` files, respectively, and the specification for these are available in the **Classes** section on **page 6** onwards.

Examples of these configuration files are available in the provided scaffold; check for the files `sample_paths.txt`, `sample_items.txt`, `sample_chasers.txt`, and `sample_exits.txt` Empty lines encountered in any configuration file can be safely skipped and ignored.

If fewer than 4 arguments are supplied, print: `"Usage: python3 game.py <PATHS> <ITEMS> <CHASERS> <EXITS>"` and exit the program.

Of the given arguments, you can expect that they will *all* be paths to a file (and/or the name of a file). **If any one of the files specified by the arguments do not exist,** print: `"You have specified an invalid configuration file."` and exit the program.

<PATHS>

The first argument, `<PATHS>`, should be the name of a file containing a list of all connections between **Locations** in the program. You can use this file to generate `Location` objects as defined on **page 6.** Each line is of the form:

```
START > DIRECTION > DESTINATION
```

Where `START` and `DESTINATION` are the names of `Location` objects, and `DIRECTION` indicates a direction on the compass (northwest, north, northeast, east, southeast, south, southwest, west) that a creature must travel towards in order to move between `START` and `DESTINATION`. For example:

```
Lake > northeast > Trail
Trail > SOUTH > Gazebo
```

If a `Creature` is currently at the `Lake`, it can move `NORTHEAST` to arrive at the `Trail`. Similarly, if a `Creature` is currently on the `Trail`, it can travel `SOUTH` in order to arrive at the `Gazebo`.

Paths do not need to be reflective - going north in location A and immediately going back south does not have to lead you back to location A. This is a game, so some paths might not make sense, and that's ok! For example:

```
Lake > NORTH > Gazebo
Gazebo > South > Gazebo
```

Location names are *case sensitive*, but `DIRECTION`s are not.

When the program starts, the player will begin in the FIRST room specified by this file. **If an empty `<PATHS>` file is given,** print: `"The game cannot run without any rooms :("` and exit the program.

You can reasonably expect that any `<PATHS>` file we will give you will contain valid input (i.e. no weird lines like `"Grass > Fence > More Grass"` or `"Not enough > arrows"`.)

`<ITEMS>`

The second argument, `<ITEMS>`, should be the name of a file defining all the `Item` objects that are available in-game. Each line is of the form:

```
short_name | item_name | full_desc | terror_rating | location
```

- Most of these are explained in the description of the **Item** class on **page 9.**
- `full_desc` is a long description of the item which is written to a `Location`'s description if the item exists at that location, but has not been picked up by a `Creature`. More information on this in can be found in the **Location** description on **page 6.**
- `location` - as it implies - specifies the name of a `Location` that the `Item` object will appear in when the game begins. You can expect that this is name will also appear somewhere in the `<PATHS>` configuration file.

For example:

```
rake | battered, old rake | A battered, old rake lies here, dreaming of lakes. | 2 |
Garden
```

It is possible for the game to start and end with no items, so it is possible that the file specified by `<ITEMS>` is empty. Should this be the case, the game should run as normal.

`<CHASERS>`

The third argument, `<CHASERS>`, should be the name of a file defining all `Creature` objects in the game that are *not* controlled by the player. Each line of this file is of the form:

```
name | description | terror_rating | location | direction
```

- The `name` of a `Creature` is a **unique** name that the program (and the user) can use to refer to the `Creature` object.
- A `description` of the `Creature` will be written to a `Location`'s description if the `Creature` is in the same `Location` as the player.
- `terror_rating` describes the terror_rating that a `Creature` begins with at the start of the game. This number represents how terrifying something is - or how difficult a `Creature` is to frighten. More details about this can be found in the **Creature** description on **page 10.**
- `location` - as it implies - specifies the name of a `Location` that the `Creature` object will appear in when the game begins. You can expect that this is name will also appear somewhere in the `<PATHS>` configuration file.
- `direction` describes a movement pattern that a `Creature` will engage in if it is not controlled by the player. More on this on **pages 10-12.**

An example:

```
dog | A dog is barking at you! | 4 | Park | North
```

As the premise of the game is not possible without any creatures to chase the player, the `<CHASERS>` file cannot be empty! **If an empty `<CHASERS>` file is provided,** print: `"There is nothing chasing you!"` and exit the program.

`<EXITS>`

The fourth argument, `<EXITS>`, should be the name of a file that contains a list of `Location` names. Each of these `Location` objects will allow the player to use the `FLEE` command (**page 16**) to win the game. If a line specifies a name that does not correspond to a `Location`, ignore it and move on to the next one.

It is possible for the `<EXITS>` configuration file to be empty - in that case, the player has no choice but to scare off all other `Creatures` to win!

Classes

Locations

Each location in the game is represented by an object of the `Location` class, which you can define in the given `location.py` scaffold. A `Location` object is characterised by:

- A `name` attribute - this is unique to each `Location`; no two `Location` objects should have the same `name`!
- The ability to **store `Item` objects**. When the game begins, a `Location` may already contain one or more `Item` objects (as dictated by the `<ITEMS>` configuration file.)
    - The player may also choose to `DROP` an object at their current location.

For each Location **`loc`**, you will also want to:

- Store all other locations that a `Creature` can reach by moving in a compass direction away from **`loc`**. (These are defined by the `<PATHS>` configuration file, written on **page 3**.)
- Know whether or not using the `FLEE` command when the player is at Location **`loc`** will end the game. (These are defined by the `<EXITS>` configuration file, written on **page 5**.)

At the start of the game, and whenever the player enters a new `Location` or whenever the `LOOK` command is invoked, a **display** representing their current `Location` is printed to standard output. Such a display consists of:

- A visualisation of the room and its possible exits. This visualisation is always **5 lines long**, and is populated by boxes `[ ]` to indicate locations reachable from the player's position. There are **nine** possible boxes total: one for each point of the compass, and one for the player's current location. The `rstrip()` method may be useful here.
    - The current location, `[x]`, is always in the middle of the **third line** (about four spaces/character widths from the left edge).

```
    [x]


You are now at: A Room With No Paths.
There is nothing here.

>>
```

- If it is possible for the player to reach another `Location` by moving **northwest** of their current position, another box `[ ]` will appear on the **top left corner** of the display, with a line leading to it from the current location `[x]` - like so:

```
[ ]
   \
    [x]


You are now at: A Room With One Path.
There is nothing here.

>>
```

- Likewise, if it is possible for the player to reach another `Location` by moving **north** of their current position, another box will appear in the middle of the **top row** of the display - also with a line leading from the current location.

```
    [ ]
     |
    [x]


You are now at: A Different Room.
There is nothing here.

>>
```

  - Similarly for the **northeast/top right corner**:

```
        [ ]
       /
    [x]


You are now at: New Room.
There is nothing here.

>>
```

  - ... and so on for all the points of the compass. Be aware that a `Location` can have multiple paths!

```
    [ ]
     |
    [x]-[ ]
   /
[ ]
You are now at: A Room With Three Paths.
There is nothing here.

>>
```

```
[ ]    [ ]
   \   /
[ ]-[x]
     |
    [ ]
You are now at: A Room With Many Paths.
There is nothing here.

>>
```

- So a room with all possible paths leading out will look as follows:

```
[ ] [ ] [ ]
   \ | /
[ ]-[x]-[ ]
   / | \
[ ] [ ] [ ]
You are now at: A Crossroads.
There is nothing here.

>>
```

- Additionally, adjacent Locations that have one or more non-player `Creature`s inside of them will be represented with `[C]` instead.

```
[C] [C] [ ]
   \ | /
[ ]-[x]-[ ]
   / | \
[C] [ ] [C]
You are now at: A Crossroads.
There is nothing here.

>>
```

- A line indicating the name of the player's current location. It follows the form: `"You are now at: <name>."`
- A paragraph of variable length, derived from the `full_desc` attributes of all `Item` objects in the current `Location`, followed by the `description` attributes of all `Creature` objects in the current `Location` that are not controlled by the player (such `Creatures` will be described in more detail on **page 10**.)

```


[ ]-[x]
     |
    [ ]
You are now at: The Den.
There is an abandoned hot dog here. Goosechaser Alice is trying to slow you down by
scattering breadcrumbs at your feet.

>>
```

    - If there are multiple `Item` objects at a `Location`, the `full_desc` of the first item to be present in the `Location` will be printed first.
    - e.g. If the `Garden` contains a knife, then the player drops a donut in the `Garden`, the `full_desc` of the knife will be printed first, then the donut.
        - If there is no way to tell which `Item` arrived at the location first (i.e. they both started in the same room), this priority instead goes to the first `Item` to be defined in the `<ITEMS>` configuration file.
    - If two or more `Creature`s not controlled by the player are in the same `Location` when this happens, the first `Creature` to appear in the `<CHASERS>` configuration file will have its `description` printed first.
    - If no `Items` or non-player `Creatures` are at the player's current `Location`, this line will instead say, `"There is nothing here."`
- If it is possible for the player to successfully use the `FLEE` command at a location (this is a win condition! Check the **FLEE** command on **page 16** for more details), the following line is added to the end of the display: `"The path to freedom is clear. You can FLEE this place."`

```
[ ]-[x]
     |
    [ ]
You are now at: The Den.
There is an abandoned hot dog here. Goosechaser Alice is trying to slow you down by
scattering breadcrumbs at your feet.
The path to freedom is clear. You can FLEE this place.

>>
```

Items

Every item in the game is represented by an object of the `Item` class, which you can define in the given `item.py` scaffold. An `Item` object has the following attributes:

- A `short_name` is an abbreviation of the item's full name, which the user can use to refer to the item when entering commands. To avoid confusion, `short_name`s are **unique** and case-insensitive.
- `item_name` is the item's name in full, usually a short description of what it is. `item_name`s are also **unique** and case-insensitive.
- `full_desc` is a long description of the item which is written to a `Location`'s description if the `Item` exists at that location but has not been picked up by a `Creature`.
- `terror_rating` describes how much more terrifying a `Creature` will look if it is carrying an item. This should be a number, and can be negative or positive. Whenever a creature picks up an item, the item's `terror_rating` is immediately added to the creature's. Whenever a creature drops an item, the item's `terror_rating` is immediately subtracted from the creature's.

Creatures

An object of the `Creature` class represents an entity in your game - you can define this class in the given `creature.py` scaffold. It could represent the goose that the player is trying to lead to safety, or any number of Goosechasers that are attempting to corner it. All non-player `Creature` objects should be defined in the `<CHASERS>` configuration file (**page 5**). `Creatures` are characterised by the following properties:

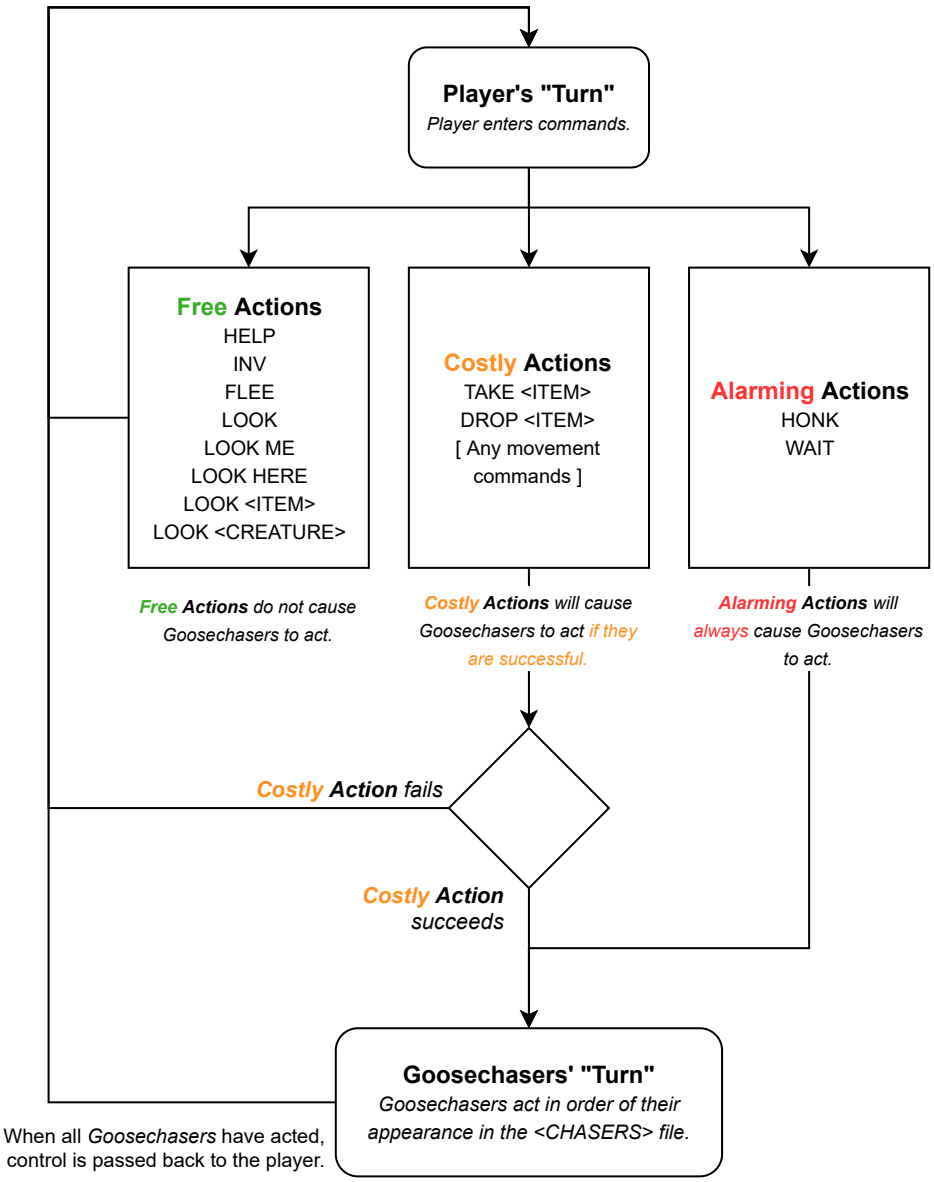
- A `name` - a **unique** and case-insensitive name that the program (and the user) can use to refer to the `Creature` object.
- A `terror_rating`. This number represents how terrifying something is - or how difficult a `Creature` is to frighten.
    - At the start of the game, the player's (goose's) `terror_rating` is **5**.
    - Every other `Creature` object's initial `terror_rating` is defined by the `<CHASERS>` file.
    - ALL `Creatures`, including those not controlled by the player, can obtain and carry `Item` objects that can modify its `terror_rating`.
- Some way of keeping track of the `Item` objects being carried by the `Creature`.
- Some way of keeping track of *where* the creature is - possibly by tying it to a `Location`. Be aware that all `Creature` objects should be able to move from one `Location` to another - and that more than one `Creature` can exist at the same `Location` at a time.

Goosechasers (Non-Player Creatures)

On top of the above, non-player `Creatures` (let's call them **Goosechasers**) might have a `description` - a short description of this entity that is written to a `Location`'s display when the player is in the same room as the `Creature`.

Goosechasers are also special because of how they move and interact with the rest of the game! Each Goosechaser has a `direction` property that tells them what direction they'll attempt to move in if they have to move. This property is defined for each Goosechaser in the `<CHASERS>` configuration file.

Every time the player **moves to a new location, uses the `WAIT` or `HONK` commands, drops an item, or picks up an item,** all Goosechasers in the game will "take their turn" - starting with the Goosechaser that appears first in the `<CHASERS>` configuration file, then the one that appears second, and so on. The player's (goose's) action - moving, waiting, or interacting with items - will always resolve *before* any Goosechasers can take their turn. A diagram has been provided on the next page to better illustrate how Goosechasers react to specific commands.



On their respective turns, a Goosechaser will perform *one* of the following actions, in order of priority:

- If a Goosechaser and the player are in the same `Location`, the Goosechaser will try to catch the goose (player).
    - If the player's `terror_rating` is not greater than that of the Goosechaser attempting to catch them, they are caught, and the game ends:

```
[ ] [ ]
   \ |
    [x]-[ ]


You are now at: The Danger Zone.
Goosechaser Bob is here!

>> DROP BALL
You drop the ball.

Bob is trying to catch you!
Oh no, you've been caught!
========= GAME OVER =========
```

    - However, if the player's `terror_rating` is *higher* than that of the Goosechaser attempting to catch them, it will take the Goosechaser *two* attempts to catch the player.
        - If a Goosechaser moves to a different room, the number of attempts required for a Goosechaser to catch the player will reset. (i.e. Goosechasers that are frightened of the player must attempt to catch the player twice *in the same room* before succeeding.)

```
[ ] [ ]
   \ |
    [x]-[ ]


You are now at: The Danger Zone.
Goosechaser Bob is here!

>> DROP BALL
You drop the ball.

Bob is trying to catch you!
But your presence still terrifies them...

>> DROP KNIFE
You drop the knife.

Bob is trying to catch you!
Oh no, you've been caught!
========= GAME OVER =========
```

- If a Goosechaser is able to reach the player's current `Location` by moving in a specific direction `D`, it will set its `direction` attribute to `D` and move that way (where `D` is one of the compass directions).
    - If a Goosechaser arrives at a `Location` while the player is still in it, the following message is printed: `"<name> has arrived at <location name>."`

```
[ ] [ ]
   \ |
    [x]-[C]


You are now at: The Danger Zone.
There is nothing here.

>> DROP BALL
You drop the ball.

Catherine has arrived at The Danger Zone.

>>
```

- If a Goosechaser is in a `Location` that contains one or more `Item` objects, it will pick up one `Item`.
    - If there is more than one `Item` object at a `Location`, the Goosechaser will take the `Item` whose `full_desc` would be printed first in the `Location`'s display.
    - If two or more Goosechasers are in the same `Location` when this happens, the first Goosechaser to appear in the `<CHASERS>` configuration file will act first.
- The Goosechaser will attempt to leave its current location through the path specified by its `direction` (e.g. if Goosechaser Bob's `direction` attribute is `"north"`, they will try to access the location to the north.)
- If the `direction` it is trying to leave in is not an available path (i.e. it does not lead to another `Location`), then it will go to the next direction that is **clockwise** from its original direction. An example:
    - Goosechaser Delila's `direction` attribute says to go `east`, but their current location only has an exit to the `south` and `north`.
    - Goosechaser Delila looks for the eastern path. There is no eastern path.
    - Goosechaser Delila changes their direction to `southeast`. There is no southeast path.
    - Goosechaser Delila changes their direction to `south`. There is a southern path. Goosechaser Delila leaves to the south.
- If their current `Location` has no paths leading away from it, the Goosechaser will do nothing.

Be aware that it is possible for Goosechasers to be **scared away permanently** (see: the `HONK` command on **page 16**). This removes the `Creature` - and any `Item`s that it is carrying - from the game entirely. Goosechasers that have been scared away by a `HONK` will no longer appear in the game: they no longer appear in location displays, do not appear in location descriptions, can no longer be `LOOK`ed at, and have no interest in hunting down the goose (player).

Commands

Unless stated otherwise, all commands are case-insensitive.

`QUIT`

At any point, the player can end the game.

```
>> QUIT
Game terminated.
```

`HELP`

The game will list all valid commands and their usage.

```
>> HELP
HELP            - Shows some available commands.
INV             - Lists all the items in your inventory.
TAKE <ITEM>     - Takes an item from your current location.
DROP <ITEM>     - Drops an item at your current location.

LOOK or L       - Lets you see the map/location again.
LOOK <ITEM>     - Lets you see an item in more detail.
LOOK ME         - Sometimes, you just have to admire the feathers.
LOOK <CREATURE> - Sizes up a nearby creature.
LOOK HERE       - Shows a list of all items in the room.

NORTHWEST or NW - Moves you to the northwest.
NORTH or N      - Moves you to the north.
NORTHEAST or NE - Moves you to the northeast.
EAST or E       - Moves you to the east.

SOUTHEAST or SE - Moves you to the southeast.
SOUTH or S      - Moves you to the south.
SOUTHWEST or SW - Moves you to the southwest.
WEST or W       - Moves you to the west.

FLEE            - Attempt to flee from your current location.
HONK or Y       - Attempt to scare off all creatures in the same location.
WAIT            - Do nothing. All other creatures will move around you.
QUIT            - Ends the game. No questions asked.

>>
```

On each line of this output, the left side before the dash is always padded so that it is **16 characters in width.**

`INV`

Prints the goose (player)'s inventory. Each item's `item_name` is printed, like so:

```
>> INV
You, a goose, are carrying the following items:
 - big knife
 - reflective sunglasses
 - prescription glasses

>>
```

If the goose is carrying exactly one item, it instead prints:

```
>> INV
You, a goose, are carrying the following item:
 - big knife

>>
```

If the user is carrying nothing, it instead says:

```
>> INV
You are carrying nothing.

>>
```

`TAKE <ITEM>`

Takes the specified item from the player's current location, and adds it to their inventory. This action can change the player's `terror_rating`.

```
[ ] [ ]
   \ |
[ ]-[x]-[ ]
       \
        [ ]
You are now at: Old Road.
Someone has stabbed a knife into the ground here. It looks rusty.

>> TAKE KNIFE
You pick up the rusty old knife.

>> INV
You, a goose, are carrying the following item:
 - rusty old knife

>>
```

If the user attempts to take something that doesn't exist, however, it prints:

```
>> TAKE CAKE
You don't see anything like that here.
```

`DROP <ITEM>`

Removes the specified item from the player's inventory, and places it at their current location. This action can change the player's `terror_rating`.

```
[ ] [ ]
   \ |
[ ]-[x]-[ ]
       \
        [ ]
You are now at: Old Road.
There is nothing here.

>> DROP knife
You drop the rusty old knife.

>> LOOK
[ ] [ ]
   \ |
[ ]-[x]-[ ]
       \
        [ ]
You are now at: Old Road.
Someone has stabbed a knife into the ground here. It looks rusty.

>>
```

If you attempt to drop something that doesn't exist in your inventory, however, the program will produce the following:

```
>> DROP CAKE
You don't have that in your inventory.
```

`FLEE`

If the player invokes this command at one of the locations specified by the `<EXITS>` configuration file, the game is won.

```
[ ]-[x]
     |
    [ ]
You are now at: The Den.
There is an abandoned burrito here. Goosechaser Alice is trying to slow you down by
scattering breadcrumbs at your feet.
The path to freedom is clear. You can FLEE this place.

>> FLEE
You slip past the dastardly Goosechasers and run off into the wilderness! Freedom at
last!
========= F R E E D O M =========
```

If the player attempts to `FLEE` at a `Location` that is not specified by the `<EXITS>` configuration file, however, the attempt fails.

```
[ ] [ ]
   \ |
[ ]-[x]-[ ]
       \
        [ ]
You are now at: Old Road.
Someone has stabbed a knife into the ground here. It looks rusty.

>> FLEE
There's nowhere you can run or hide! Find somewhere else to FLEE.

>>
```

`HONK` or `Y`

Attempts to scare off every other creature at the player's location using the player's `terror_rating`. If the player's `terror_rating` is higher than that of a target creature's, they are scared away permanently!

```
[ ]-[x]
     |
    [ ]
You are now at: The Den.
There is an abandoned burrito here. Goosechaser Alice is trying to slow you down by
scattering breadcrumbs at your feet. A dog is minding its own business here.

>> HONK
You sneak up behind your quarry and honk with all the force of a really angry airhorn!
HONK!
Alice is spooked! They flee immediately!
Dog is spooked! They flee immediately!

>> LOOK


[ ]-[x]
     |
    [ ]
You are now at: The Den.
There is an abandoned burrito here.

>>
```

If you successfully scare away ALL the creatures in the game, you win (as illustrated on the next page.)

```
[ ]-[x]
     |
    [ ]
You are now at: The Den.
There is an abandoned burrito here. Goosechaser Alice is trying to slow you down by
scattering breadcrumbs at your feet. A dog is minding its own business here.

>> Y
You sneak up behind your quarry and honk with all the force of a really angry airhorn!
HONK!
Alice is spooked! They flee immediately!
Dog is spooked! They flee immediately!

None can stand against the power of the goose!
========= V I C T O R Y =========
```

If your `terror_rating` is less than or equal to that of a creature you honked at, the creature is not frightened - and will probably try to catch you.

```
[ ]-[x]
     |
    [ ]
You are now at: The Den.
There is an abandoned burrito here. Goosechaser Alice is trying to slow you down by
scattering breadcrumbs at your feet. A dog is minding its own business here.

>> Y
You sneak up behind your quarry and honk with all the force of a really angry airhorn!
HONK!
Alice is not spooked :(
Dog is spooked! They flee immediately!

Alice is trying to catch you!
Oh no, you've been caught!
========= GAME OVER =========
```

If you `HONK` when no other creatures are in the room with you:

```
[ ] [ ]
   \ |
    [x]-[ ]


You are now at: The Danger Zone.
There is nothing here.

>> Y
All shall quiver before the might of the goose! HONK!

>>
```

`WAIT`

This causes all `Creature`s that are not controlled by the player (i.e. Goosechasers) to "take their turn" (pick up an item, move one room, attempt to catch the goose, etc.)

```
[ ] [ ]
   \ |
    [x]-[C]


You are now at: The Danger Zone.
There is nothing here.

>> WAIT
You lie in wait.

Catherine has arrived at The Danger Zone.

>> LOOK
[ ] [ ]
   \ |
    [x]-[ ]


You are now at: The Danger Zone.
Goosechaser Catherine is brandishing a net at you, threateningly.

>>
```

`LOOK` and `L`

Displays the location you are currently in.

```
>> LOOK


[ ]-[x]
       \
        [ ]
You are now at: The Shallows.
There is nothing here.

>>
```

```
>> L


[ ]-[x]
       \
        [ ]
You are now at: The Shallows.
There is nothing here.

>>
```

`LOOK <ITEM>`

Allows the player to examine items in their inventory or at their current location. The item's `short_name` will work for this. The output for this command follows this format: `"<item_name> - Terror Rating: <terror_rating>"`, as shown below.

```
>> INV
You, a goose, are carrying the following items:
 - big knife
 - reflective sunglasses
 - prescription glasses

>> LOOK KNIFE
big knife - Terror Rating: 5

>>
```

```
[ ]-[x]
     |
    [ ]
You are now at: The Den.
There is an abandoned taco here.

>> LOOK TACO
really sad taco - Terror Rating: 1

>>
```

If no such item exists, print the following; note that the item name given is converted to lower case.

```
>> LOOK BURGER
You don't see anything like that here.

>>
```

`LOOK ME`

Displays your current stats.

```
>> LOOK ME
You are a goose. You are probably quite terrifying.
In fact, you have a terror rating of: 5

>>
```

`LOOK <CREATURE>`

Assesses a creature and how frightened it seems of the goose (player). Uses the creature's `name` as the argument. The creature must be in the same location as the player. Measures the player's `terror_rating` against target creature's. If the player's `terror_rating` exceeds the target creature's by **5 or more,** then the result looks like this:

```
        [ ]
       /
[ ]-[x]
   /   \
[ ]     [ ]
You are now at: The Burrows.
Dog is here, for some reason.

>> LOOK DOG
Dog looks a little on-edge around you.
```

If the target creature's `terror_rating` exceeds that of the player's by **5 or more**, then the result instead looks like this:

```
        [ ]
       /
[ ]-[x]
   /   \
[ ]     [ ]
You are now at: The Burrows.
Dog is here, for some reason.

>> LOOK DOG
Dog doesn't seem very afraid of you.
```

In all other cases, print the following:

```
        [ ]
       /
[ ]-[x]
   /   \
[ ]     [ ]
You are now at: The Burrows.
Cat is here, for some reason.

>> LOOK CAT
Hmm. Cat is a bit hard to read.
```

If there is no such `Creature` present at the player's `Location`, this action fails.

```
[ ]-[x]-[ ]
       \
        [ ]
You are now at: Under the Moonlight.
There is nothing here.

>> LOOK CAT
You don't see anything like that here.
```

LOOK HERE

Lists the `short_name`s and `item_name`s of all `Item` objects at the player's current `Location`.

```
    [ ]
     |
    [x]-[ ]
   /
[ ]
You are now at: a Big Tree.
There is a rake on the ground, dreaming of lakes. Red and unblemished, a freshly-fallen
apple rests on the ground here. A weird blue rock minds its own business in the corner,
unmoving. There is a bucket of gold paint here. It is dangerously full.

>> LOOK HERE
RAKE            | ordinary rake
APPLE           | freshly-fallen apple
ROCK            | blue rock
BUCKET          | bucket of paint

>>
```

The `short_name` of each item is always displayed in **upper-case letters**. The left side is also padded so that it is always **16 characters in width.**

If there are no `Item` objects at the player's `Location`, it instead prints:

```
>> LOOK HERE
There is nothing here.
```

Movement Commands

Moves the user to a connecting location in that specified direction.

```
        [ ]
       /
[ ]-[x]
   /   \
[ ]     [ ]
You are now at: The Burrows.
Cat is here, for some reason.

>> SW
You move southwest, to The Granary.
    [ ] [ ]
     | /
    [x]-[ ]
     |
    [ ]
You are now at: The Granary.
There is nothing here.

>>
```

If there is no room that can be accessed by moving in the specified direction, the program will instead print:

```
>> SOUTHEAST
You can't go that way.
```

Invalid Commands

If the user enters an invalid command, print `You can't do that.` and ask for another command.

```
>> SLEEP
You can't do that.

>>
```

Sample Output

Here is a sample output of the program, which has been run using the sample configuration files given to you in the scaffold. Note that the program begins by printing a display and description of the first location to appear in the `<PATHS>` configuration file, then by prompting the user for a command.

```
[ ]     [C]
   \   /
[ ]-[x]
     |
    [ ]
You are now at: Fountain.
A grimy copper coin is stuck fast to the ground here. A weird blue rock minds its own
business in the corner, unmoving.

>> look here
COIN            | strange copper coin
ROCK            | blue rock

>> LOOK coin
strange copper coin - Terror Rating: 2

>> look ME
You are a goose. You are probably quite terrifying.
In fact, you have a terror rating of: 5

>> east
You can't go that way.

>> cry
You can't do that.

>> take coin
You pick up the strange copper coin.

Dog has arrived at Fountain.

>> look dog
Hmm. Dog is a bit hard to read.

>> northwest
You move northwest, to Phone Booth.

Dog has arrived at Phone Booth.
        [ ]
       /
    [x]
   /   \
[C]     [ ]
You are now at: Phone Booth.
Bright red and striking, a ribbon adds a bit of colour to the ground here. A dog is
barking at you!

>> ne
You move northeast, to Painted Fences.
```

```
Dog has arrived at Painted Fences.
    [ ] [ ]
     | /
    [x]-[ ]
   /
[ ]
You are now at: Painted Fences.
There is a bucket of green paint here. It is dangerously full. A dog is barking at you!

>> wait
You lie in wait.

Dog is trying to catch you!
But your presence still terrifies them...

>> n
You move north, to Windmill.

Dog has arrived at Windmill.


    [x]-[ ]
     |
    [ ]
You are now at: Windmill.
Someone has stabbed a knife into the ground here. It looks rusty. A dog is barking at
you!
The path to freedom is clear. You can FLEE this place.

>> take knife
You pick up the rusty old knife.

Dog is trying to catch you!
But your presence still terrifies them...

>> look me
You are a goose. You are probably quite terrifying.
In fact, you have a terror rating of: 12

>> look dog
Dog looks a little on-edge around you.

>> HONK
You sneak up behind your quarry and honk with all the force of a really angry airhorn!
HONK!
Dog is spooked! They flee immediately!

Greg has arrived at Windmill.

>> INV
You, a goose, are carrying the following items:
 - strange copper coin
 - rusty old knife

>> FLEE
You slip past the dastardly Goosechasers and run off into the wilderness! Freedom at
last!
```

```
========= F R E E D O M =========
```

Submission and Mark Breakdown

Submit your assignment on **Ed** in the *Assignments* section of the **Assessments** tab. Your final submission is due on **Monday, November 4th 2019 AEST (Week 13 Monday).** The marking breakdown of the assignment is as follows (**10 marks**):

- **3 marks** will be awarded as a progress mark, as described in the *Milestone Submission* section below.
- **2 marks** will be awarded for code correctness, assessed by automatic test cases on Ed. Some test cases will be hidden and will not be available before the deadline.
- **3 marks** will be awarded through manual marking. Your tutors will check your code for:
  - Code style and readability
  - Appropriateness of code comments
  - Code and logical structures
- A further **2 marks** will be awarded for the submission of a **test driver program** (this is separate from the test cases submitted as part of your milestone submission). Such a program will be described in further detail on **page 29**.
  - **1 mark** will be awarded for the presence and correctness of test cases submitted.
  - **1 mark** will be awarded for the justification of these test cases.

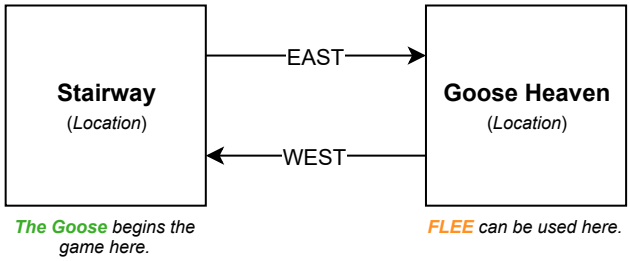
Milestone Submission

2 marks can be awarded for the code submitted in a milestone submission (here) before **Monday, October 28th 2019 AEST (Week 12 Monday)**. These marks will be awarded based on the number of automatic test cases passed, which will generally test for the following:

- The completion (and correctness) of the functions in `preprocessing.py`
  - For the purposes of this milestone submission, please complete the functions in `preprocessing.py` as accurately to the scaffold as you can; down to appropriate `return` values (where applicable).
- Handling missing arguments
- Handling missing files
- Program start-up (first display + input loop)
- Working input loop
  - Handling invalid commands
- The following commands:
  - `HELP`
  - `QUIT`
  - `LOOK ME` when no items are carried
  - `INV` when no items are carried

Additionally, **1 mark** will be awarded for submitting a suite of *four test cases*. These test cases must be available in a separate `tests` directory that is included in your **last submission** to the milestone assessment, alongside your actual program.

We will describe a scenario for each of these test cases. Your task is to provide the relevant **configuration files** for each test case, and the **expected output** of the program. All of the test cases described will have the following `Location` structure:



Scenario 1: No Goosechasers

Configuration Files: [`1_paths.txt`, `1_items.txt`, `1_chasers.txt`, `1_exits.txt`]

The **expected output** of your program should be written in another file, labelled `1_expected.txt`.

In this scenario, no creatures are chasing after the goose (player). No inputs will be given to your program.

Scenario 2: One Goosechaser, Goose Flees

Configuration Files: [`2_paths.txt`, `2_items.txt`, `2_chasers.txt`, `2_exits.txt`]

The **expected output** of your program should be written in another file, labelled `2_expected.txt`.

In this scenario, there is *one* Goosechaser that begins the game at the exit `Location`, **Goose Heaven**. Upon execution, the program will receive the following inputs:

```
EAST
FLEE
```

The goose (player)'s attempt to flee will be successful.

Scenario 3: One Goosechaser, Goose Caught

Configuration Files: [`3_paths.txt`, `3_items.txt`, `3_chasers.txt`, `3_exits.txt`]

The **expected output** of your program should be written in another file, labelled `3_expected.txt`.

In this scenario, there is *one* Goosechaser that begins the game at the exit `Location`, **Goose Heaven**. Upon execution, the program will receive the following inputs:

```
EAST
```

The goose will be caught, and the game will end.

Scenario 4: One Goosechaser, Goose Indomitable

Configuration Files: [`4_paths.txt`, `4_items.txt`, `4_chasers.txt`, `4_exits.txt`]

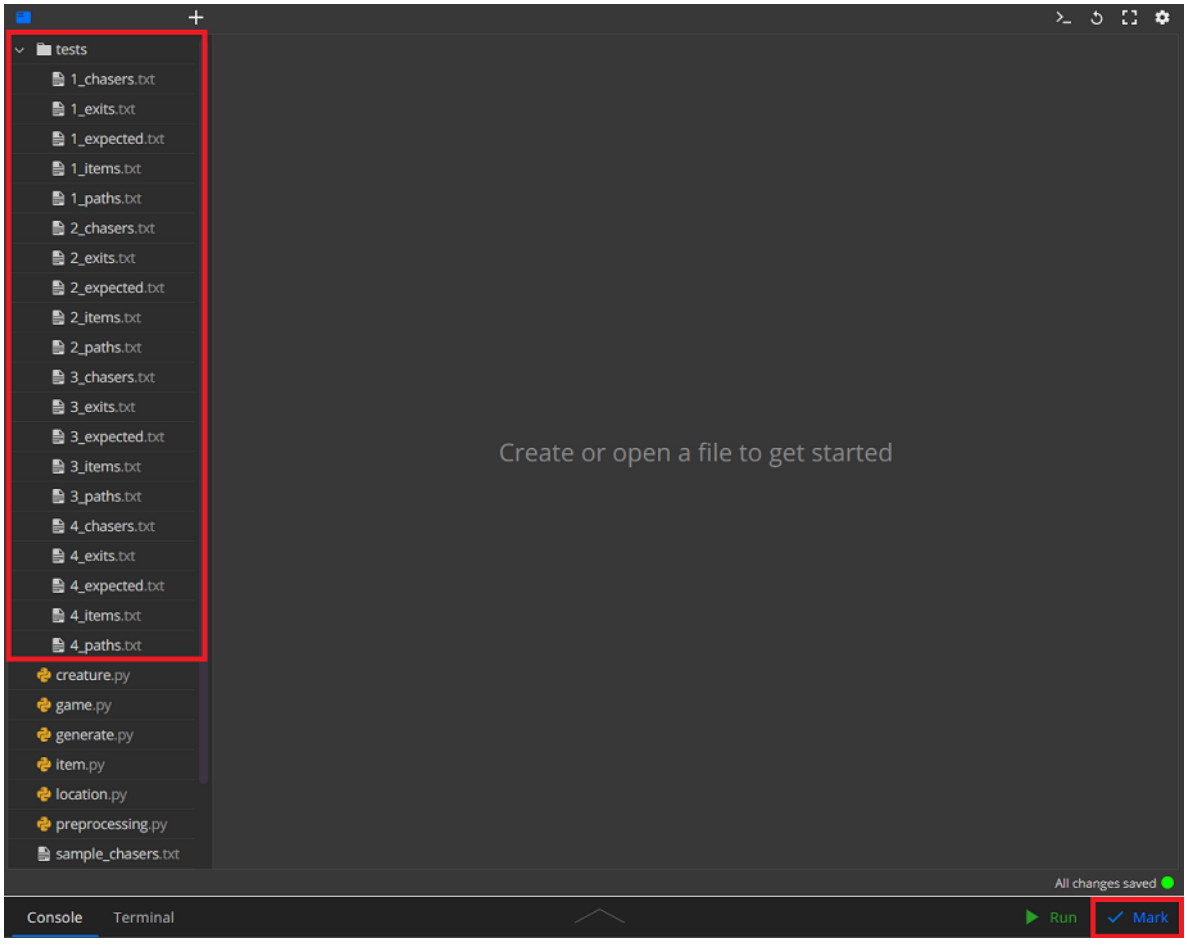
The **expected output** of your program should be written in another file, labelled `4_expected.txt`.

In this scenario, there is *one* Goosechaser that begins the game at the exit `Location`, **Goose Heaven**. Upon execution, the program will receive the following inputs:

```
EAST
HONK
```

The goose will succeed in scaring away the Goosechaser, and the game will end.

Uploading Your Test Cases



Remember to upload all your correctly-labelled test case files in a separate `tests` directory alongside your milestone submission. It is not sufficient to simply upload the test cases to your workspace: for your tutor to be able to access these files, you need to click the **Mark** button as well - even if you've already received all the green ticks to confirm that you've passed the automatic test cases.

Final Submission: Test Cases

As previously described, **2 marks** will be awarded for the submission of a **test driver program** - which is basically just a program that runs some unit tests for you. Alongside your final submission, you will submit an additional Python program, titled `creature_test.py`. This will be a test driver program for the `Creature` class defined on **page 10** of this document.

The program `creature_test.py` will define and run **at least 5** different unit tests for the `Creature` class.

- Each test must include the creation of a `Creature` object, and use any combination of the constructor (`__init__`) and the methods `take()`, `drop()`, and `get_terror_rating()`.
- Each test will be assigned some number `X`. Whenever a test fails, it prints the message: `"Test X failed."`, where `X` is the number it has been assigned.
- The description and data for each test case must be written **as comments** within `creature_test.py`. No other files should be required for unit testing.
    - The mark awarded for the justification of your test cases will be assessed through this component of your submission.

A template for this `creature_test.py` has been included as part of your scaffold. If you're unsure of where to start on this, reviewing components of the course that touch on the concept of testing is a good idea.

Academic declaration

By submitting this assignment, you declare the following:

I declare that I have read and understood the University of Sydney Student Plagiarism: Coursework Policy and Procedure, and except where specifically acknowledged, the work contained in this assignment/project is my own work, and has not been copied from other sources or been previously submitted for award or assessment.

I understand that failure to comply with the Student Plagiarism: Coursework Policy and Procedure can lead to severe penalties as outlined under Chapter 8 of the University of Sydney By-Law 1999 (as amended). These penalties may be imposed in cases where any significant portion of my submitted work has been copied without proper acknowledgment from other sources, including published works, the Internet, existing programs, the work of other students, or work previously submitted for other awards or assessments.

I realise that I may be asked to identify those portions of the work contributed by me and required to demonstrate my knowledge of the relevant material by answering oral questions or by undertaking supplementary work, either written or in the laboratory, in order to arrive at the final assessment mark.

I acknowledge that the School of Computer Science, in assessing this assignment, may reproduce it entirely, may provide a copy to another member of faculty, and/or communicate a copy of this assignment to a plagiarism checking service or in-house computer program, and that a copy of the assignment may be maintained by the service or the School of Computer Science for the purpose of future plagiarism checking.