EECS3311 Software Design Fall 2020
Project
Designing and Implementing the Space Defender 2 Game

Chen-Wei Wang and Kevin Banh

Released Date: Monday, November 2
Phase 1 Due Date (Sections A & E): **11:59pm (EST), Wednesday, November 25**
Phase 2 Due Date (Sections A & E): **11:59pm (EST), Wednesday, December 9**

Policies

- **Your (submitted or un-submitted) solution to this project (which is not revealed to the public) remains the property of the EECS department. Do not distribute or share your code in any public media (e.g., a non-private Github repository) in any way, shape, or form. The department reserves the right to take necessary actions upon found violations of this policy.**
  - You are required to **work on your own** for this project. **No** group partners are allowed.
  - When you submit your project, you claim that it is **solely** your work (both the software and the design document). Therefore, it is considered as **an violation of academic integrity** if you copy or share **any** parts of your Eiffel code or the design document during **any** stages of your development.
  - When assessing your submission, the instructor and TA will examine your code, and suspicious submissions will be reported to the department/faculty if necessary. **We do not tolerate academic dishonesty**, so please obey this policy strictly.
- You are entirely responsible for making your submission in time.
  - You are encouraged to submit **multiple times** prior to the deadline: only the last submission before the deadline will be graded.
  - Practice submitting your project early **even before it is in its final form**.
  - No excuses will be accepted for failing to submit shortly before the deadline.
  - Back up your work **periodically**, so as to minimize the damage should any sort of computer failures occur. Follow this tutorial series on setting up a **private** Github repository for your Eiffel projects.
  - The deadline is **strict** with no excuses: you receive **0** for not making your electronic submission in time.
  - **Emailing your solutions, or links to a cloud drive, to the instruction or TAs will not be accepted.**
- You are free to work on this lab on your own machine(s), but you are responsible for testing your code at a remote Prims lab machine before the submission.

Contents

1 Individual Work

As already specified and explained in your course syllabus, only individual work is allowed for the project. Here are the rationales:

1. Given the inability to gather physically, we want to minimize the disadvantage of those of you who cannot find a suitable team member and/or who cannot collaborate effectively virtually (over different time zones).

   Furthermore, we would like to avoid you encountering issues which are (much) more likely to occur or even exacerbated in a completely online context: time is wasted between coordinating meeting times and deliverables among members, there is an unfair distribution of work between members, and/or a contributing member withdraws from the course at the last moment or becomes non-responsive.

2. We reckon that individual work is the best way for you to learn (and achieve the CLOs) and for us to assess your understanding and design/implementation skills.

3. Knowing that this course is run completely online this term, this project was developed over the past summer with the intention for a single student to complete over the five-week period. We also release more starter tests to guide your development. Moreover, as you can see from the suggested work sequence in the project instruction, the design problem may be solved most effectively by implementing modules sequentially, rather than being coordinated between members.

4. Compared with projects in the previous semesters, the current project is made lighter in its scale and difficulty. But of course it still requires solid work and a steady development process.

We will be actively holding office/TA hours and moderating the course forum to help you complete the project. Please start early with your development and approach us early.

2 Cross References to Lab3 Instructions

To avoid repetitions (and thus satisfy the **single choice principle**), here are sections which you would still consult with from your Lab3 instructions:

- `Working on Your Own Machine`
- `Grading Criterion of this Lab`: The grading criterion for the software part of your project (both Phase 1 and Phase 2) will be consistent with that for your Lab3: each acceptance test is considered as **passing** only if the output generated by your program is character-by-character identical to that generated by the oracle. **Therefore, it is crucial that you constantly run regression tests on your software.**
- `Reporting Issues of the Oracle`
- `Task 1 of Lab3: Required Tutorials` (Review how to use ETF if necessary.)
- `Developing your ETF Project: GUI Mode vs. Command-Line Mode`
- `Modification of the Cluster Structure`

3 Grading Scheme for Phase 1

The sole purpose of the Phase 1 submission (2% of your course grade) is to ensure that you are making reasonable progress, so as to minimize the need to rush through your development for Phase 2. See Section 8.1 for details on the submission. The mark you get is based on the number of starter tests your submission passes. There are 15 tests provided in the starter folder. Out of the 2%, you will receive:

- 2% if your submission passes at least ($\geq$) 11 starter tests
- 1.5% if your submission passes at least ($\geq$) 9 starter tests
- 1% if your submission passes at least ($\geq$) 7 starter tests
- 0% if your submission passes less than 7 starter tests.

4 Suggested Sequence of Development for this Project

– Review how to use ETF if necessary.

– Spend a good amount of time reading through the requirements of the game:

- Section 6
- Text file `messages.txt` (messages for abstract state)
- Text file `errors.txt` (messages for errors)
- Example acceptance tests: `at001.txt` to `at015.txt`, and their corresponding expected files.

  Any clarifications needed for these requirements items, post your questions on the course forum.

– Implement the game:

- Start by implementing everything outside of the `in game` state.
- Implement the initial state of the `in game` state.
- Implement the Starfighter's movement and `pass` command.
- Implement the friendly projectiles.
- Add the enemies.
- Add the scoring system.
- Implement the Starfighter's specials.
- Throughout the above steps, create acceptance tests and test against the oracle.

– Write the design document (Section 7).

5 Getting Started

– First of all, make sure you have already acquired the basic knowledge about the Eiffel Testing Framework (ETF) which can be found in the tutorial videos and parts of Lecture Series 6.

– Download the **`system_space_defender_2.zip`** file from the course eClass page and unzip it.

– **Follow this tutorial video to get started from the downloaded starter file**:

`https://www.youtube.com/watch?v=pjXjlPNpb5c&list=PL5dxAmCmjv_5q9wBFXV-M4S4VbB1T0EJg&index=9`

– **In the starter folder, there is a small project `starter_code`, which consists of two classes**

- `RANDOM_GENERATOR`
- `RANDOM_GENERATOR_ACCESS`

which you are simply expected to re-use *verbatim*, when implementing the two generated numbers as described in Section 6.13.7, in the `model` cluster of your project.

6 Task 1 (Phase 1 & 2): Development of the Space Defender 2 Game

6.1 (Informal) Background

Many years have passed since the first battle between the creatures of the Void and humankind. Time changes everything. With the research branch at work, the current iteration of **Starfighter** feature customizable parts for weapons, armour and engine, allowing for specialization based on the battlefield. In addition, salvaged parts (orbment and focus) from the enemy units have allowed for development of special abilities for Starfighters. The countless engagements with the enemy monsters allowed for a better understanding of them. So far, five different types of enemy units have been discovered. First is the generic Grunt. Next is the improved version of a Grunt, the Fighter. We have the large Carrier which releases a torrent of drones known as Interceptor. The Interceptor's goal is to simply absorb projectile damage, serving as a shield to other units. Finally, a massive structure called Pylon which heals surrounding units. The enemy units appear to behave differently based on how the Starfighter acts and whether they see the Starfighter or not. Your goal is to update the simulation made in the previous lab to accommodate the new technologies and information. Unfortunately, time travelling is not one of the special abilities developed for the Starfighter hence there is no need to keep the undo/redo from the previous simulation.

6.2 Contrast with Lab3

This project asks you to develop a game independent of Lab3, though it has some limited extent of similarity with the simple game you developed for Lab3. To understand the rules of the game, it is important that you read carefully the rest of this section. Also note that for this project, you do not need to implement the undo/redo design pattern.

Section 4 suggests sequence of developments. However, you need to constantly run regression tests on your software for incremental developments.

6.3 Vocabulary

- Orbment: Can be either a singular orb or a container (focus). Enemies drop orbments when they are destroyed, and orbment has a different value (score).
- Focus: A container for orbments. Starfighter has one focus with an unlimited capacity. On the other hand, the focus an enemy drops has a limited capacity, and its score is amplified when filled (see Section 6.15 for how scoring works).

6.4 Looks of Abstract State

In this section we present some abstract state displays that your design/implementation must support:

- When the software is first launched, we are in the `not started` state (i.e., the game is not yet started):

```
state:not started, normal, ok
Welcome to Space Defender Version 2.
```

- Upon using the `play` command, the software goes to the setup stage. The setup stage is composed of 5 states: four of these states are for the Starfighter to select different parts (e.g., weapon, armour), and most states look similar to below. Commands `setup_back` and `setup_next` are used to navigate between states in the setup stage, whereas command `setup_select` is used to select options.

  Commands `setup_back` and `setup_next` allow you to move back and forth between the five setup state: if you setup back too far, you will end up in the `not started` state (as if you invoked the `abort` command); if you setup next too far, you will end up starting the game. See Section 6.6 for more details on the setup stage.

```
->play(5,10,1,1,1,1,1)
  state:weapon setup, normal, ok
  1:Standard (A single projectile is fired in front)
    Health:10, Energy:10, Regen:0/1, Armour:0, Vision:1, Move:1, Move Cost:1,
    Projectile Damage:70, Projectile Cost:5 (energy)
```

```
  2:Spread (Three projectiles are fired in front, two going diagonal)
    Health:0, Energy:60, Regen:0/2, Armour:1, Vision:0, Move:0, Move Cost:2,
    Projectile Damage:50, Projectile Cost:10 (energy)
  3:Snipe (Fast and high damage projectile, but only travels via teleporting)
    Health:0, Energy:100, Regen:0/5, Armour:0, Vision:10, Move:3, Move Cost:0,
    Projectile Damage:1000, Projectile Cost:20 (energy)
  4:Rocket (Two projectiles appear behind to the sides of the Starfighter and accelerates)
    Health:10, Energy:0, Regen:10/0, Armour:2, Vision:2, Move:0, Move Cost:3,
    Projectile Damage:100, Projectile Cost:10 (health)
  5:Splitter (A single mine projectile is placed in front of the Starfighter)
    Health:0, Energy:100, Regen:0/10, Armour:0, Vision:0, Move:0, Move Cost:5,
    Projectile Damage:150, Projectile Cost:70 (energy)
  Weapon Selected:Standard
```

– At the end of the setup stage, a summary of the chosen parts for the Starfighter will be displayed.

```
->play(5,10,1,1,1,1,1)
  ... -- first setup state omitted
->setup_next(4) -- move 4 states forward from the current setup state
  state:setup summary, normal, ok
  Weapon Selected:Standard
  Armour Selected:None
  Engine Selected:Standard
  Power Selected:Recall (50 energy): Teleport back to spawn.
```

– At any point in time, the two toggle commands may be used. The key command out of those two is `toggle_debug_mode` (which you need to implement to switch the debug mode on or off). The other command `toggle_RNG_out` is not required to be implemented by you.

As an example, here is the output when you invoke `toggle_debug_mode` outside of a game:

```
->toggle_debug_mode
  state:not started, debug, ok
  In debug mode.
```

As another example, here is the output when you invoke `toggle_debug_mode` when a game has been started:

```
->toggle_debug_mode
  state:in game(0.1), debug, ok
  In debug mode.
```

– When a game has been started, its state is displayed differently according to whether debug mode is on (see Section 6.14.1 for more details). When the debug mode is *on*, more information is displayed. Here is an example game state where the debug mode is *off*:

```
->setup_next(5)
  state:in game(0.0), normal, ok
  Starfighter:
    [0,S]->health:70/70, energy:70/70, Regen:1/3, Armour:1, Vision:13, Move:10, Move Cost:3, location:[E,1]
      Projectile Pattern:Standard, Projectile Damage:70, Projectile Cost:5 (energy)
      Power:Recall (50 energy): Teleport back to spawn.
      score:0
      1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
    A _  _  _  _  _  _  _  _  _  _  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?
    B _  _  _  _  _  _  _  _  _  _  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?
```

```
   C _  _  _  _  _  _  _  _  _  _  _  _  _  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?
   D _  _  _  _  _  _  _  _  _  _  _  _  _  _  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?
   E S  _  _  _  _  _  _  _  _  _  _  _  _  _  _  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?
   F _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?
   G _  _  _  _  _  _  _  _  _  _  _  _  _  _  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?
   H _  _  _  _  _  _  _  _  _  _  _  _  _  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?
   I _  _  _  _  _  _  _  _  _  _  _  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?
   J _  _  _  _  _  _  _  _  _  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?
```

Here is another example game state where the debug mode is *on*:

```
->setup_next(5)
  state:in game(0.0), debug, ok
  Starfighter:
    [0,S]->health:70/70, energy:70/70, Regen:1/3, Armour:1, Vision:13, Move:10, Move Cost:3, location:[E,1]
      Projectile Pattern:Standard, Projectile Damage:70, Projectile Cost:5 (energy)
      Power:Recall (50 energy): Teleport back to spawn.
      score:0
  Enemy:
  Projectile:
  Friendly Projectile Action:
  Enemy Projectile Action:
  Starfighter Action:
  Enemy Action:
  Natural Enemy Spawn:
      1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
    A _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _
    B _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _
    C _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _
    D _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _
    E S  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _
    F _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _
    G _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _
    H _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _
    I _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _
    J _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _
```

– The only two commands that can be invoked any time are `toggle_debug_mode` and `toggle_RNG_out`. All other commands must be used with constraints (see Section 6.14.2); failing them will lead to error messages. For example:

```
->abort
  state:not started, normal, error
  Command can only be used in setup mode or in game.
```

6.5 Some Commands to Know

– In order to begin the process of setting up the Starfighter's equipments, the user first invokes the `play` command, e.g., `play(10,30,1,1,1,1,1)`. After the setup process is completed, we say that a game of an indefinite number of turns begins. Now we discuss the seven argument values of the `play` command:

- The first two numbers are for specifying the size of the board (in this example, a 10-by-30 board). The first number represents the number of rows (ranging from 5 to 10), whereas the second number represents the number of columns (ranging from 10 to 30).
- Each of the remaining five numbers ranges from 1 to 101 and must be non-decreasing (e.g., invoking `play(_, _, 101, 101, 60, 60, 1)` should trigger an error message). The combination of these last 5 numbers serves the following purpose: during the end of each turn in a game, a random number generator (RNG) is used to generate two numbers (and we refer to the first number as i and the second number as j).

  Say the number of rows is r and the number of columns is c.

  ◇ The first number i (ranging from 1 to r) determines which row in the last column of the board an enemy should spawn (i.e., an enemy should spawn at location [i,c]).

◇ The second number (ranging from 1 to 100) is used to determine which enemy spawns:

◇ From an invocation of the play command `play(_,_,`$n1$`,`$n2$`,`$n3$`,`$n4$`,`$n5$`)`, the last five numbers specify the **thresholds** for Grunt ($n1$), Fighter ($n2$), Carrier ($n3$), Interceptor ($n4$), and Pylon ($n5$). And these five threshold values suggest six intervals to consider for generating enemies (to see more about this random generation of enemies, see Section 6.13.7):

· If the second number j is in the interval $[1, n1)$ (i.e., between 1 inclusive and the grunt threshold $n1$ exclusive), then a Grunt will spawn.

· If the second number j is in the interval $[n1, n2)$ (i.e., between grunt threshold inclusive and fighter threshold exclusive), then a Fighter will spawn.

... (This pattern continues.)

· If the second number j is is in the interval $[n5, 101)$ (i.e., between pylon threshold inclusive and 101 exclusive), then nothing is spawned.

◇ As an example, consider the invocation of `play(6,25,60,60,101,101,101)`:

If the first number i generated is 5, then the enemy spawn location would be [E,25]. (Recall that rows are represented by letters, and E is 5th letter in the alphabet.) Here the first interval is $[1, 60)$, meaning that if the second number j is between 1 and 59, a Grunt will spawn. The second interval is $[60, 60)$ which is empty, meaning that no Fighter can ever spawn. The third interval is $[60, 101)$, meaning that if the second number is between 60 and 100, a Carrier will spawn. The fourth and fifth intervals are both empty (i.e., $[101, 101)$), meaning that no Interceptor and Pylon can ever spawn. Finally, the sixth interval is also empty (i.e., $[101, 101)$), meaning that in each turn, an enemy (i.e., Grunt or Carrier) will always spawn.

◇ As another example, consider the invocation of `play(10,30,1,1,1,1,1)`:

If the first number i generated is 10, then the enemy spawn location would be [J,30]. Given that the first five intervals are all empty (i.e., $[1, 1)$), none of the five kinds of enemies can ever spawn in each turn. The fact that the sixth interval is "universal" (i.e., $[1, 101)$) makes sure that the second number j (ranging from 1 to 100) generated will always fall within this interval and does not spawn any enemy. That is, regardless of the second number, based on the rules above, no enemies will spawn.

Remark. When you test the correctness of your design, you may try various combinations of these threshold values.

– The `abort` command can be used to exit from either the setup stage (before a game starts) or the current turn (during a game).

– The `toggle_debug_mode` command "flips" the debug mode (e.g., to *on* if it is currently *off*). By default, the debug mode is initialized to *off*. If in game mode and the debug mode is *on*, the abstract state displayed shows more information. Essentially, a normal game should have its debug mode switched *off*, whereas switching it *on* shows everything in that state which allows for a programmer to debug.

When the debug mode is switched *off*, only the following is displayed:

- basic state information (like state number)
- information about the Starfighter
- a board with information limited by the fog of war mechanics

  (i.e., the Starfighter has limited vision on locations on the board; see **Vision** in Section 6.6)
- whether the game is over

For more information, refer to Section 6.14.

– `toggle_RNG_out`: This command is available for use in the oracle provided, but is not required for you to implement (see Section 6.16 for more details). The purpose is to help you debug RNG (random number generator) usage. This command can be used anytime and toggles a flag on use. If invoked in a game, the flag is set to true and the effect will only take place in the next turn. More precisely, in the next turn, there will be an additional `RNG Usage:` section appended at the end displaying the two randomly generated numbers (i.e., the row number and the enemy value) for spawning enemies.

6.6 Setup

The software, once launched, has a number of *states* for its progression. The logical sequence is you begin in a `not started` state, move to the setup stage (which itself consists of five states) before moving into a `in game` state. In the following, we describe each of these states.

1. `not started` state: you arrive in this state as soon as you launch the software (e.g., in the interactive mode via `oracle.exe -i`).

2. setup stage: Invoking the `play` command moves the player from the `not started` state to the beginning of the setup stage. The setup stage consists of five states, `weapon setup`, `armour setup`, `engine setup`, `power setup`, `setup summary`.

   – Navigating between each state in the setup stage is done via the `setup_back` and `setup_next` commands. Each of these commands has an integer parameter between 1 and 5. Using the `setup_back` or `setup_next` command moves the state from the current state, respectively, backward or forward by the specified integer amount of states.

   Consequently, going before the `weapon setup` state (e.g., invoke `setup_back(2)` when the current state is `armour setup`, the 2nd setup state) will lead you to back to the `not started` state. Similarly, going beyond the `setup summary` state (e.g., invoke `setup_next(4)` when the current state is `armour setup`) will lead you to the `in game` state.

   For example, say after invoking the `play` command initially, we invoke the following in sequence: `setup_next(2)`, `setup_back(1)`, and `setup_next(5)`. What will be the intermediate states?

   - We start at `weapon setup` due to the `play` command.
   - By invoking `setup_next(2)`, we move to the `engine setup` state, which is two states ahead of `weapon setup`.
   - By invoking `setup_back(1)`, we move back one state which is `armour setup`.
   - By invoking `setup_next(5)`, given that `setup summary` is 3 states ahead of the current state (`armour setup`), we go beyond the `setup summary` state and arrive in the `in game` state.

   It is also possible to invoke the `abort` command to exit from the setup stage and go back to the `not started` state.

   – Except for the `setup summary` state, each of the other four states of the setup stage allows the player to choose the equipments (i.e., weapon, armour, engine, and power) for the Starfighter. By default, option 1 is selected for the weapon, armour, engine, and power. By invoking the `setup_select` command. The `setup_select` command takes as input an integer between 1 and 5 and selects the corresponding option displayed for that particular state in the setup stage.

   Selected equipments are preserved; you can overwrite the selected equipments by invoking the `setup_select` command; the history of selected equipments is wiped out only when the software is terminated. This means if a certain equipment is chosen and the setup stage is exited[1], the next time the setup stage is entered, that certain equipment will be selected by default.

   The `setup summary` summarizes what equipments the Starfighter is entering the game with.

   There are 5 different types of weapons, 4 different types of armours, 3 different types of engines, and 5 different types of powers. Weapons, armours, and engines share some common attributes (see below). Attribute values for each weapon, armour, and engine, along with the error message for selecting an invalid equipment option, can be found in Section 6.14. Additional information about weapons and powers can be found in Section 6.7 and Section 6.8, respectively.

   The Starfighter's final attribute values, upon entering the `in game` state, are the sum of its parts. Attributes which weapon, armour, and engine share in common include:

   - `Health`: It represents the maximum integrity of the entity (e.g., Starfighter, Grunt), also known as the total health. An entity's current health value has the lower bound of 0 and upper bound of this maximum `Health` value. If the current health falls to 0, the entity is destroyed. Note that in the game, for the Starfighter only, it is possible for its current health value to exceed this maximum `Health` value by using a power.

[1] Recall there are three ways for exiting the setup stage: `abort`, `setup_back` before the `weapon` state, or `setup_next` beyond the `setup_summary` state.

- `Energy`: It represents the maximum energy of the Starfighter, also known as the total energy. Note that the current energy cannot exceed total energy (with the exception of using a power). Energy is required for the Starfighter to move, to use a power, or to fire projectiles.
- `Regen` (for regeneration): There are two numbers.
  - ◇ The first number represents the amount of current health the Starfighter gains per turn passively (**health regeneration**). Recall that the current health cannot exceed total health, so regeneration only works up to the total health value (except with the use of a power).
  - ◇ The second number is for **energy regeneration** and it works the same way as health (with respect to the `Energy` value above).

  Note. `Regen` is the base value for the (health or energy) regeneration process to take place.
- `Armour`: If the Starfighter and a projectile collide, the damage done to the Starfighter is the projectile's damage minus the Starfighter's armour. Armour does not have any cancelling effect on collisions with non-projectiles.
- `Vision`: It represents how many spaces away, on the board, the Starfighter can see from its current location.

  For example, if the Starfighter is in [D,1], in normal mode, to see what entity is in [C,3], you need a vision of 3 (1 vertical + 2 horizontal) or more. Spaces beyond the range of the Starfighter's vision are known to be in the *fog of war* and, in normal mode, are represented by a `?` symbol on the board output.

  Note that in debug mode, the *fog of war* is not displayed on the board output.
- `Move`: It represents how many spaces away the Starfighter can move from its current location (assuming it has enough energy). The distance calculation is the same as vision calculation: vertical distance plus horizontal distance in terms of spaces.
- `Move Cost`: It represents how many energy unit it takes for the Starfighter to move 1 space. If the move cost is 5 and we make a move from [A,1] to [D,3], it would cost 25 energy units: 5 spaces (3 vertical + 2 horizontal) $\times$ 5.

3. `in game` state: this is when the game has begun. See later sections.

6.7 Weapon Types

The weapon chosen plays a role in how the Starfighter's `fire` command affects the board when used in game. Each weapon has the shared attributes described above (e.g., health, energy). In addition, each weapon has other attributes that determine: **1)** the health or energy cost to fire the projectiles; and **2)** the base damage of the projectiles (when they collide with other entities). These attributes can also be found in Section 6.14; to complement that section, here we describe:

- When the Starfighter fires, how many projectiles spawn and where they spawn.
- After spawning, assuming the **friendly** projectiles are still on the board, every subsequent turn later, how the **friendly** projectiles act also depends on the weapon selected.

There are 5 different types of weapons.

- `Standard`: One projectile is spawned. It appears to the right of the Starfighter. For example, if the Starfighter is at [A,2], the projectile appears at [A,3]. Every subsequent turn after, the projectile moves 5 units to the right.
- `Spread`: Three projectiles are spawned:
  - The first one is spawned to the top-right of the Starfighter, the second to the right of the Starfighter while the last projectile is spawned to the bottom-right of the Starfighter. For example:
  - If the Starfighter is at [D,4], the projectiles appear at [C,5], [D,5] and [E,5] in that order.
  - Every subsequent turn after, the projectiles move 1 unit in the direction where they spawned relative to the Starfighter when it fired. For example, the projectile at [C,5] will move up right to [B,6] while the projectile at [D,5] will move to [D,6].

    Note. Recall how the Starfighter travels diagonally: vertical first, then horizontal. Contrast this with how a projectile travels diagonally (which is a real diagonal movement). For example, if there was an entity at [B,5], no collision will occur from the projectile travelling from [C,5] to [B,6]. On the other hand, if the Starfighter was to travel from [C,5] to [B,6], it would first travel vertically to [B,5], resulting in a collision.

- `Snipe`: One projectile is spawned. It appears to the right of the Starfighter. For example, if the Starfighter is at [A,2], the projectile appears at [A,3]. Every subsequent turn after, the projectile moves 8 units to the right.

  Note. Unlike all the other projectiles, this projectile travels by "teleporting" to the final location. More precisely, if there are any entities along the horizontal path to the targeted location, the projectile will not collide with any of those entities unless there is an entity in the targeted location (which is 8 units to the right of the initial location).

- `Rocket`: Two projectiles are spawned. The first one is spawned to the top-left of the Starfighter while the second projectile is spawned to the bottom-left of the Starfighter. For example, if the Starfighter is at [B,2], the projectiles appear at [A,1] and [C,1] in that order. Every subsequent turn after, the projectiles move starting at 1 unit to the right and **doubling** (because it's a rocket!) in movement each turn.

  Note. If the Starfighter is in the first column and they fire, the projectiles will be spawned outside the board behind them, in which case we still assign ids to them. However, in the next turn, since these projectiles are already outside the board, we do not keep track of them, meaning that they will not move into the board.

- `Splitter`: One projectile is spawned. It appears to the right of the Starfighter. For example, if the Starfighter is at [A,2], the projectile appears at [A,3]. The projectile does not move.

6.8 Specials

There are 5 different kinds of special/power a Starfighter may have. Powers may be used in game via the Starfighter's `special` command. The cost, type of cost, and a brief description of each kind of power can be found in Section 6.14. Below is a more in-depth description of each power.

- `Recall`: When you start a game, the Starfighter spawns in a certain location (see Section 6.10). When this special is used, teleport the Starfighter to that location. Unlike moving, you can still use this command if the Starfighter is to be teleported to the same location. Make sure to account for collisions at the spawn location.

- `Repair`: Increase current health by 50 units.

  Note. this is the only case that current health may exceed the total health for a Starfighter.

- `Overcharge`: This is to sacrifice current health for gaining current energy. This is dependent on the amount of the current health spent. It will use up to 50 current health as long as the Starfighter is not destroyed to convert to current energy. The conversion is: 1 current health for 2 current energy.

  For examples: If you have 1 current health, 0 current health will be converted to 0 current energy. If you have 11 current health, 10 current health will be converted to 20 current energy, and there is still 1 current health left. If you have 100 current health, 50 current health (hence the up to) will be converted to 100 current energy, and there is still 50 current health left.

  Note. This is the only case that current energy may exceed total energy for a Starfighter.

- `Deploy Drones`: Clears out all the projectiles on the board. The order of removal is from projectiles that spawned earliest to latest.

- `Orbital Strike`: All enemies take 100 damage, reduced by their armour value. Suppose an enemy has 15 armour, they will take 85 (i.e., 100 - 15) damage. The order of taking damage is from enemies that spawned earliest to latest.

6.9 Enemies

There are various kinds of enemies (described), and they have shared attributes, including:

- health (total health)

- regen (health regeneration per turn)

- armour (which mitigates damage: one point of damage to one point of armour for the collision with a friendly projectile or an Orbital Strike by the Starfighter)

- vision (which works the same way as the Starfighter's vision)

- `seen_by_Starfighter` (whether or not the enemy is within the Starfighter's vision; only updated in certain phases of a turn; see Phase 4 and Phase 6 in Section 6.13)
- `can_see_Starfighter` (whether or not the Starfighter is within the enemy's vision; only updated in certain phases of a turn; see Phase 4 and Phase 6 in Section 6.13)

Enemies may also *preempt* certain moves depending on the Starfighter's actions, and act differently depending on whether they see the Starfighter or not. See Section 6.13 for more details on preempt. Furthermore, enemies spawn other entities, and spawned entities do not act on the turn they are spawned. Entities that spawn outside the board must have an **id** associated with them, but there is no need to keep track of them. Here are the kinds of enemies with their initial attribute values and how they act:

1. `Name:Grunt`, Health:100, Regen:1, Armour:1, Vision:5.
   - `Preemptive Action`: If the Starfighter passes, increase both current health and total health by 10. If the Starfighter uses a special, increase current health and total health by 20. Turn does not end in both cases.
   - `Action when Starfighter is not seen`: Moves 2 spaces left. If the Grunt is still on the board and is not destroyed, fire a projectile which spawns directly to the left of the Grunt. The spawned projectile moves left 4 spaces per turn and has a base damage of 15.
   - `Action when Starfighter is seen`: Moves 4 spaces left. If the Grunt is still on the board and is not destroyed, fire a projectile which spawns directly to the left of the Grunt. The spawned projectile moves left 4 spaces per turn and has a base damage of 15.
2. `Name:Fighter`, Health:150, Regen:5, Armour:10, Vision:10.
   - `Preemptive Action`: If the Starfighter fires, increase armour by 1 and do not end the turn. If the Starfighter passes, move 6 spaces left. If the Fighter is still on the board and is not destroyed, fire a projectile which spawns directly to the left of the Fighter. The spawned projectile moves left 10 spaces per turn and has a base damage of 100. End the Fighter's turn.
   - `Action when Starfighter is not seen`: Moves 3 spaces left. If the Fighter is still on the board and is not destroyed, fire a projectile which spawns directly to the left of the Fighter. The spawned projectile moves left 3 spaces per turn and has a base damage of 20.
   - `Action when Starfighter is seen`: Moves 1 space left. If the Fighter is still on the board and is not destroyed, fire a projectile which spawns directly to the left of the Fighter. The spawned projectile moves left 6 spaces per turn and has a base damage of 50.
3. `Name:Carrier`, Health:200, Regen:10, Armour:15, Vision:15.
   - `Preemptive Action`: If the Starfighter uses a special, increase regen by 10 and do not end the turn. If the Starfighter passes, move 2 spaces left. If the Carrier is still on the board and is not destroyed, spawn two Interceptors: one directly above it and then one directly below it. End the Carrier's turn.

     Note. If an enemy is occupying a spawn location for the Interceptor, do not spawn the Interceptor. If the spawn location is outside the board, treat it similar to a projectile spawning outside of the board. This means spawn the Interceptor (including assigning an ID to it, which will be discussed later), but do not keep track of that entity as it is out of the board.
   - `Action when Starfighter is not seen`: Moves 2 spaces left.
   - `Action when Starfighter is seen`: Moves 1 space left. If the Carrier is still on the board and is not destroyed, spawn an Interceptor directly left of the Carrier.
4. `Name:Interceptor`, Health:50, Regen:0, Armour:0, Vision:5.
   - `Preemptive Action`: If the Starfighter fires, **attempt** to move the Interceptor vertically directly to the row the Starfighter is in before ending the Interceptor's turn. More precisely about the vertical movement:
     - **Case 1.** If the Interceptor and the Starfighter are in different columns, then:
       - ◇ If the path to the row where the Starfighter is in is clear, then then move to that row (without colliding with the Starfighter).
       - ◇ If there is at least one occupying enemy blocking the path, move as close to the first one (so as to avoid a collision). Along the way to this first occupying enemy, the Interceptor may collide with projectiles.

◇ If there is no occupying enemy blocking the path, but there is one or more occupying projectiles, then the Interceptor collides with those projectiles.

- **Case 2.** If the Interceptor and the Starfighter are in the same column, then:
  - ◇ If the path to the Starfighter's location is clear, then collide with the Starfighter.
  - ◇ If there is at least one occupying enemy blocking the path, move as close to the first one (so as to avoid a collision). Along the way to this first occupying enemy, the Interceptor may collide with projectiles.
  - ◇ If there is no occupying enemy blocking the path, but there is one or more occupying projectiles, then the Interceptor collides with those projectiles before it may collide with the Starfighter.

Note for Case 1 and Case 2. The Interceptor may be destroyed along the path, due to collisions with projectiles, in which case it is just removed from the board without continuing moving along the path.

– `Action when Starfighter is not seen`: Moves 3 spaces left.

– `Action when Starfighter is seen`: Moves 3 spaces left.

5. `Name:Pylon`, Health:300, Regen:0, Armour:0, Vision:5.

– `Preemptive Action`: None.

– `Action when Starfighter is not seen`: Moves 2 spaces left. If the Pylon is still on the board and is not destroyed, then for each enemy within the Pylon's vision range (including itself), heal it for 10 current health. Note that current health cannot exceed total health when healing this way.

– `Action when Starfighter is seen`: Moves 1 space left. If the Pylon is still on the board and is not destroyed, fire a projectile which spawns directly to the left of the Pylon. The spawned projectile moves left 2 spaces per turn and has a base damage of 70.

6.10 Initial Game State

The initial board output when the game is started in normal (and not debug mode) is displayed below:

```
   1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
A  _  _  _  _  _  _  _  _  _  _  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?
B  _  _  _  _  _  _  _  _  _  _  _  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?
C  _  _  _  _  _  _  _  _  _  _  _  _  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?
D  _  _  _  _  _  _  _  _  _  _  _  _  _  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?
E  S  _  _  _  _  _  _  _  _  _  _  _  _  _  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?
F  _  _  _  _  _  _  _  _  _  _  _  _  _  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?
G  _  _  _  _  _  _  _  _  _  _  _  _  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?
H  _  _  _  _  _  _  _  _  _  _  _  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?
I  _  _  _  _  _  _  _  _  _  _  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?
J  _  _  _  _  _  _  _  _  _  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?
```

In this example:

– There are 10 rows and 30 columns, meaning that the `play` command used must have been invoked as something like: `play(10,30,?,?,?,?,?)`.

– The Starfighter's vision is 13 as any location beyond 13 spaces away (sum of vertical and horizontal displacement from [E,1]) is covered by the fog of war (which is represented by `?`), denoting the fact the entity there is unknown.

– If debug mode was on, clearing all of the fog of war, every location except [E,1] would be denoted as an empty space (_).

– The Starfighter does not always spawn at [E,1] all the time. Spawn is dependent on the number of rows specified for the board. Similar to Lab3, the Starfighter will always spawn in column 1. The row it spawns in is the ceiling of the number of rows divided by 2. The Starfighter's attributes are the sum of the parts chosen for it.

6.11 Symbols

Below are a list of symbols that may appear on the board.

- `?`: This symbol only shows up in normal mode. It represents the fog of war meaning from the perspective of the Starfighter, the location is too far away for the Starfighter to see what entity is in that location.
- `_` : This symbol means the location is not occupied by any entity.
- `G` : This symbol means the location is occupied by a Grunt.
- `F` : This symbol means the location is occupied by a Fighter.
- `C` : This symbol means the location is occupied by a Carrier.
- `I` : This symbol means the location is occupied by a Interceptor.
- `P` : This symbol means the location is occupied by a Pylon.
- `<` : This symbol means the location is occupied by an enemy projectile (fired by some kinds of enemies).
- `S` : This symbol means the location is occupied by the Starfighter (which fires friendly projectiles).
- `*` : This symbol means the location is occupied by a friendly projectile.
- `X` : This symbol means the location is where the Starfighter was destroyed. This symbol is only present when the game is over due to destruction of the Starfighter.

6.12 Entity IDs

Each entity has a unique integer ID associated with it.

- The Starfighter has an ID of 0.
- Projectiles have an ID starting at -1, decreasing by 1 for each new (enemy or friendly) projectile spawned.
- Enemies have an ID starting at 1, increasing by 1 for each new enemy spawned.
- IDs are reset between games so the counters for IDs start at 1 and -1 respectively for the next game.

6.13 A Turn of the Game

A turn occurs when the `move`, `pass`, `fire` or `special` command is invoked successfully (with no errors) when in game. See Section 6.14.2 for conditions where an error will occur for each command. A turn comprises of multiple phases.

6.13.1 Phase 1: Friendly Projectiles Act

- Friendly projectiles (fired by the Starfighter) on the board move one by one, in the order of oldest projectile to newest (in terms of when the projectile was spawned).
- Movement of each friendly projectile is determined by the weapon of the Starfighter, see Section 6.7.

  Note. For the `Snipe` weapon, projectiles teleport directly to the destination, rather than travelling there.
- Possible interactions between friendly projectiles and other entities include:
  - **Colliding with an enemy projectile**: There are three cases to consider. First, if the enemy projectile's damage is higher, the friendly projectile is removed from the board. The new enemy projectile's damage is its damage subtracted by the friendly projectile's damage. Second, if the friendly projectile's damage is higher, the enemy projectile is removed from the board. The new friendly projectile's damage is its damage subtracted by the enemy projectile's damage. The friendly projectile continues its travel to its end destination. Third, if damages of both projectiles are the same, they are both removed from the board.
  - **Colliding with a friendly projectile**: The stationary projectile is removed from the board. The moving friendly projectile's damage gets added with the damage of the stationary projectile. The moving friendly projectile continues its travel to its end destination.

- **Colliding with an enemy**: The friendly projectile is removed from the board. Say the friendly projectile has damage d, and the enemy has current health h and armour value a. The enemy's current health is updated to: $h - max(d - a,\ 0)$. That is, when $a \geq d$, there is no harm to the current health. If enemy's current health becomes either 0 or below, then after the collision, the enemy is destroyed and removed from the board.
- **Colliding with the Starfighter**: The friendly projectile is removed from the board. Say the friendly projectile has damage d, and the Starfighter has current health h and armour value a. The Starfighter's current health is updated to: $h - max(d - a,\ 0)$. That is, when $a \geq d$, there is no harm to the current health. If the Starfighter's current health becomes 0 or below after the collision, the Starfighter is destroyed and the game is over. In this case, set the Starfighter's current health to 0, skip newer friendly projectiles' actions, and skip the rest of the turn. Change the Starfighter's symbol on the board to an `X` and output the state when the Starfighter got destroyed.

6.13.2 Phase 2: Enemy Projectiles Act

– Enemy projectiles (fired by certain kinds of enemies) on the board move one by one, in the order of oldest projectile to newest (in terms of when the projectile was spawned).

– Movement of each enemy projectile is determined by the kind of enemy which fired it.

Note. All enemy projectiles travel to their end destination rather than teleporting there directly.

– Possible interactions between enemy projectiles and other entities include:

- **Colliding with an friendly projectile**: There are three cases to consider. First, if the friendly projectile's damage is higher, the enemy projectile is removed from the board. The new friendly projectile's damage is its damage subtracted by the enemy projectile's damage. Second, if the enemy projectile's damage is higher, the friendly projectile is removed from the board. The new enemy projectile's damage is its damage subtracted by the friendly projectile's damage. The enemy projectile continues its travel to its end destination. Third, if damages of both projectiles are the same, they are both removed from the board.
- **Colliding with an enemy projectile**: The stationary projectile is removed from the board. The moving enemy projectile's damage gets added with the damage of the stationary projectile. The moving enemy projectile continues its travel to its end destination.
- **Colliding with an enemy**: The enemy projectile is removed from the board. The enemy is healed: its current health is increased by the enemy projectile's damage value. Note that armour does not affect the amount healed. If the enemy's current health is higher than its total health after the heal, set its current health to the total health.
- **Colliding with the Starfighter**: The enemy projectile is removed from the board. Say the enemy projectile has damage d, and the Starfighter has current health h and armour value a. The Starfighter's current health is updated to: $h - max(d - a,\ 0)$. That is, when $a \geq d$, there is no harm to the current health. If the Starfighter's current health becomes 0 or below after the collision, the Starfighter is destroyed and the game is over. In this case, set the Starfighter's current health to 0, skip newer enemy projectiles' actions, and skip the rest of the turn. Change the Starfighter's symbol on the board to an `X` and output the state when the Starfighter got destroyed.

6.13.3 Phase 3: Starfighter Act

First, let us define the notion of **regeneration**. The Starfighter has a regen attribute denoted by two numbers (e.g., `1/2`): when regeneration occurs, there is health and energy regeneration. For health regeneration, if the current health of the Starfighter is less than the total health, add the first number of the regen attribute to the current health. If current health exceeds total health after the previous step, set current health to total health. If initially, the current health exceeds total health via special usage, do not apply health regeneration. Energy regeneration works the same way, except that it is denoted by the second number of the regen attribute. The Starfighter acts according to the command being invoked by the user:

– `pass`: Apply regeneration twice. The Starfighter does not do anything.

– `move`: Apply regeneration. Subtract the cost of moving from energy. The energy cost is the move cost multiplied by the amount of spaces between the current location and the end location (e.g., to move from [A,2] to [B,6], there are 1 + 4 = 5 spaces).

Note. The "end location" does not necessarily mean the specified location, but where the Starfighter stops. The Starfighter may be destroyed on its way to the specified location.

Similar to Lab 3, the Starfighter travels vertically then horizontally to the targeted location, interacting with other entities on the way. If the Starfighter's current health is 0 at any point, the Starfighter is destroyed and the game is over, meaning that the Starfighter does not continue to move to the targeted destination, and the remaining phases for the turn are skipped. Change the Starfighter's symbol on the board to an `X` and output the state when the Starfighter got destroyed.

Possible interactions between the Starfighter and other entities include:

- **Colliding with an enemy or a friendly projectile**. The projectile is removed from the board. Say the projectile has damage d, and the Starfighter has current health h and armour value a. The Starfighter's current health is updated to: $h - max(d - a,\ 0)$.
- **Colliding with an enemy**. The enemy is destroyed and removed from the board. Say the enemy has current health eh, and the Starfighter has current health sh. The Starfighter's current health is updated to: $sh - eh$ (i.e., armour plays no role).

For both of the above cases of collision, if the Starfighter's current health becomes 0 or below after the collision, the Starfighter is destroyed and the game is over. In this case, set the Starfighter's current health to 0 and skip the rest of the turn. Change the Starfighter's symbol on the board to an `X` and output the state when the Starfighter got destroyed.

– `fire`: Apply regeneration. Subtract the cost of firing, and spawn (friendly) projectiles based on the weapon the Starfighter is equipped with (see Section 6.7). The order of the projectile spawn is determined by the row they spawn in: spawning the projectiles closer to row `A` first (equivalently, spawning the projectiles from top to bottom of the board).

Possible interactions between the friendly projectiles spawned and other entities include:

- **Spawning in a location that has an enemy projectile**. There are three cases to consider. First, if the enemy projectile's damage is higher, the friendly projectile is removed from the board. The new enemy projectile's damage is its damage subtracted by the friendly projectile's damage. Second, if the friendly projectile's damage is higher, the enemy projectile is removed from the board. The new friendly projectile's damage is its damage subtracted by the enemy projectile's damage. Third, if damages of both projectiles are the same, they are both removed from the board. For the first and third cases, even though the friendly projectile is removed from the board after spawning, an ID needs to be associated with it.
- **Spawning in a location that has a friendly projectile**. The projectile already in that location is removed from the board. The new spawning friendly projectile's damage is added by the damage of the removed projectile.
- **Spawning in a location that has an enemy**. The friendly projectile is removed from the board. Say the friendly projectile has damage d, and the enemy has current health h and armour value a. The enemy's current health is updated to: $h - max(d - a,\ 0)$. That is, when $a \geq d$, there is no harm to the current health. If enemy's current health becomes either 0 or below, then after the collision, the enemy is destroyed and removed from the board. Even though the friendly projectile is removed from the board after spawning, an ID needs to be associated with it.
- Though not an interaction, if the friendly projectile spawns outside of the board, you don't have to keep track of it, but it needs an ID for outputting messages.

– `special`: Apply regeneration. Subtract the cost for using the special. What happens depends on the special the Starfighter has. Section 6.8 already explains what happens for each special.

6.13.4 Phase 4: Enemy Vision Update

– Update the `seen_by_Starfighter` and `can_see_Starfighter` attributes for each enemy that is still on the board:

- For `seen_by_Starfighter`, set to true if the sum of the horizontal and vertical displacement between the enemy's location and the Starfighter's location is less than or equal to the Starfighter's vision. Set to false otherwise.

- For `can_see_Starfighter`, set to true if the sum of the horizontal and vertical displacement between the enemy's location and the Starfighter's location is less than or equal to the enemy's vision. Set to false otherwise.

6.13.5 Phase 5: Enemies Act

– Recall Section 6.9 for details about various kinds of enemies.

– Depending on the Starfighter's action, certain enemies may preempt their actions.

- All the preempted actions for the enemies on the board go first. The order is from oldest to newest spawned enemies on the board.
- There are two possible consequences of each preempted action. First, a preempted action may not end the enemy's turn, meaning it can act again later. Second, a preempted action may cause the enemy's turn to end, which works the same way as a normal non-preempted action does (**see the next point**).

– After the preempted actions, all enemies on the board whose turn did not end will act from the order from oldest to newest spawned on the board.

How an enemy acts depends on whether they see the Starfighter or not and the type of the enemy they are. An enemy's action can be generalized as the following sequence of actions:

1. **Regeneration**, which works the same way as the Starfighter's regeneration. The only difference being is that there is no energy regeneration for an enemy.
2. **Movement**. The enemy will try to move to the targeted location.
   - If the path they are going through contains another enemy, the enemy will stop 1 space before hitting that enemy, while still interacting with other entities the same way as enemy interactions when spawning in Section 6.13.7 on the way there.
   - Otherwise, travel through the path, interacting with other entities the same way as enemy interactions when spawning in Section 6.13.7, stopping only when the enemy is destroyed (and removed from the board) or is at the targeted location. There is no penalty if the enemy escapes (move out of the board), and such an escaping enemy is not considered to be destroyed for scoring purposes.
3. If the enemy is still on the board, they may either **fire or spawn enemies**. Recall that when an enemy fires, it spawns enemy projectiles.
   - Enemy projectile spawning interacts with the other entities the same way as described in Section 6.13.2 except the targeted location is where they spawn.
   - Enemy spawning interacts with the other entities the same way as described in Section 6.13.7. An ID still needs to be assigned if the spawn location is outside the board. Similar to enemies naturally spawning, an enemy will not spawn in a location occupied by another enemy. Spawned enemies do not act the turn they are spawned.

Note. If the Starfighter's current health becomes 0 at any point, the Starfighter is destroyed and the game is over. In this case, set the Starfighter's current health to 0, finish the current enemy's action, skip other newer enemies' actions, and skip the rest of the turn. Change the Starfighter's symbol on the board to an `X` and output the state when the Starfighter got destroyed.

6.13.6 Phase 6: Enemy Vision Update

– Same as Phase 4. Note that these attributes **ARE NOT** modified at all in Phase 5 after each enemy acts for simplicity.

6.13.7 Phase 7: Enemy Spawn

– The RNG (provided in your starter code) is always used twice:

- The first number is generated from 1 to the number of rows on the board, which determines where the enemy spawn location will be. The spawn location is [first number, number of columns on the board].
- Recall the threshold values supplied from the `play` command used to initiate the game (see Section 6.5). The second number is generated from 1 to 100, determining what type of enemy spawn or whether if any enemy even spawns at all.

Note. If the spawn location is occupied by an enemy, nothing is spawned (meaning there is no need to create an ID for it). The RNG is still used twice though (it is just that the second number is unused).

– Possible interactions between the enemy spawned and other entities include:

- **Spawning in a location occupied by a friendly projectile**. The friendly projectile is removed from the board. Say the friendly projectile has damage d, and the enemy has current health h and armour value a. The enemy's current health is updated to: $h - max(d - a,\ 0)$. That is, when $a \geq d$, there is no harm to the current health. If enemy's current health becomes either 0 or below, then after the collision, the enemy is destroyed and removed from the board.
- **Spawning in a location occupied by an enemy projectile**. The enemy projectile is removed from the board. The enemy is healed: its current health is increased by the enemy projectile's damage value. Note that armour does not affect the amount healed. If the enemy's current health is higher than its total health after the heal, set its current health to the total health.
- **Spawning in a location occupied by the Starfighter**. The enemy is destroyed and removed from the board. Say the enemy has current health eh, and the Starfighter has current health sh. The Starfighter's current health is updated to: $sh - eh$ (i.e., armour plays no role). If the Starfighter's current health becomes 0 or below after the collision, the Starfighter is destroyed and the game is over. In this case, set the Starfighter's current health to 0. Change the Starfighter's symbol on the board to an `X` and output the state when the Starfighter got destroyed.

Note that if the enemy is immediately destroyed, an ID still needs to be assigned to it.

6.13.8 How Can A Game be Over?

– When in game, the way a game can be over is if the user uses the `abort` command.

– Another way is if the Starfighter's current health is reduced to 0 and thus destroyed.

Note. Unlike the previous lab, when the Starfighter is destroyed, the game completely stops and outputs the current state, rather than continuing the rest of the current phase and the rest of the turn (though there is an exception to this: see Phase 5 above, where the current enemy still has to finish its turn). Due to this, it is possible that `seen_by_Starfighter` and `can_see_Starfighter` attributes of enemies become inaccurate as the vision update phases have not been done.

6.13.9 How Can A Game be Won?

You cannot win a game. A user can play indefinitely as long as the game is not over.

6.14 Messages

In this section, we discuss all possible messages which your implemented game may output.

6.14.1 Abstract State Messages

See file `messages.txt`.

6.14.2 Error Messages

See file `errors.txt`.

6.15 Scoring

At the end of each turn, an updated score should be output. For example, when first entering the game:

```
->setup_next(5)
  state:in game(0.0), normal, ok
  Starfighter:
    [0,S]->health:70/70, energy:70/70, Regen:1/3, Armour:1, Vision:13, Move:10, Move Cost:3, location:[E,1]
      Projectile Pattern:Standard, Projectile Damage:70, Projectile Cost:5 (energy)
```

```
Power:Recall (50 energy): Teleport back to spawn.
score:0
```

That is, think of the score as an attribute of the Starfighter. For examples, `at010.expected.txt` and `at011.expected.txt` show some simple cases of calculating the scores (and you are expected to test the more complex cases against the oracle).

In a turn, whenever an enemy gets destroyed, the player gains points. The scoring system is based on two types of orbments: orbs and focuses. The scoring system consists of three parts: components of scoring, a mechanism for combining components, and an algorithm for calculating scores:

– **Scoring Components**

- A Grunt, on destruction, drops a silver orb worth 2 points. A Fighter drops a gold orb worth 3 points. An Interceptor drops a bronze orb worth 1 point.
- **Focuses** are containers of orbments and if they are full, a score multiplier is applied to the sum of the score of the orbments the focus is holding.
  - ◇ A Carrier drops a diamond focus which has a gold orb in the first slot. A diamond focus has the capacity of 4 orbments and if it is full, a $\times 3$ (triple) score multiplier is applied to the sum of the score of the 4 orbments.
  - ◇ A Pylon drops a platinum focus which has a bronze orb in the first slot. A platinum focus has the capacity of 3 orbments and if it is full, a $\times 2$ (double) score multiplier is applied to the sum of the score of the 3 orbments.
  - ◇ A Starfighter has its own focus that has unlimited capacity (hence there is not a multiplier). Orbments obtained from destroying other enemies are added to it.

– **Combining Scoring Components**

- In order to add an orbment to a focus (whose capacity may or may not be fixed), traverses, from left to right, as follows:
  1. If the current slot contains an orb, then skip (as an orbment cannot be added to an orb) and move on to the next slot.
  2. If the current slot contains a **focus** where you can add an orbment, then add it there; otherwise, skip and move on to the next slot.
  3. If the current slot is empty/unfilled, then add it there.
- Whenever an enemy is destroyed, add the dropped orbment to the Starfighter's focus, by applying the above traversal.

– **Calculating Scores**

- To calculate the score, traverse the Starfighter's focus:
  - ◇ Orbs are worth the flat amounts as mentioned above.
  - ◇ The score of a focus is the sum of the score of each orbment in the focus. If every slot in that focus is occupied (even by another focus that is not itself full), apply the score multiplier to the sum of the score of each orbment in the focus.

  Note. Adding an orbment and calculating the score are recursive.
- There is no need to worry about integer overflow.
- Score is not preserved between games.

6.16 Debugging Random Number Generator

- A RNG is used in game to spawn enemy units (and note that you are already provided the implemented classes `RANDOM_GENERATOR` and `RANDOM_GENERATOR_ACCESS`).
- **The `oracle.exe` program you are given supports the command `toggle_RNG_out`, which allows you to see the expected numbers to be generated each turn (so that you can debug your program's output if necessary).**
  - The command can be used anytime.
  - It toggles a flag.
  - If that flag is set to true (by default, the flag starts at false), when in game (and the state where the Starfighter is destroyed), the output has an additional section called `RNG Usage:` appended at the end.
  - If the section is empty, it means the RNG was not used.
  - **If the RNG is used in a turn, it is used twice no matter what. The first number is row number, generated from 1 to the number of rows in the board, signifying which row an enemy may spawn in. The second number is a number from 1 to 100, determining which enemy is spawning.**
- **This `toggle_RNG_out` command is not available in your ETF project. You do not need to implement it (and we will not test it).**

7 Task 2 (Phase 2): Design Document

Instructions about your project design will be made available **by the due date of Phase 1**.

8 Submission

8.1 Phase 1 Submission

8.1.1 Checklist before Submission

1. The design document is **not** needed for Phase 1 submission.
2. Make sure the *ROOT* class in the *root* cluster has its *switch* feature defined as:

```
switch: INTEGER
    -- Running mode of ETF application
  do
-- Result := etf_gui_show_history. -- GUI mode
   Result := etf_cl_show_history
-- Result := unit_test             -- Unit Testing mode
  end
```

8.1.2 Submitting Your Work

– Electronic Submission

1. You are expected to submit from a Prism lab terminal.
2. Produced outputs by your program must be **identical** to those produced by the oracle. You are responsible for testing enough input cases with the oracle give to you.
3. There are two `space_defender_2` directories: one is the top-level directory that contains all generated ETF code and your development; the other is the sub-directory that contains the code of your model of space defender. **When you submit, make sure you submit the top-level `space_defender_2` directory.**
4. **Go to the directory containing the top-level `space_defender_2` project directory**:

   4.1 Run the following command to remove the `EIFGENs` directory:

   ```
   eclean space_defender_2
   ```

   4.2 Run the following command to make your submission:

   ```
   submit 3311 Project space_defender_2
   ```

   - Here is a short tutorial video guiding you the process of submitting your labs/project from home (It is just an example for Lab0, and you would need to adapt it to Project).
   - Submissions via Web Submit or email attachments are **unacceptable** and will **not** be graded.
   - After you submit, there will be some automated program attempting to perform some **basic checks** on your program: if your submitted directory has the expected structure, if Eiffel project compiles, and etc. Please be patient and wait until it finishes.
   - **You are encouraged to submit multiple times before the deadline (this also allows you to backup your work on the EECS server).** Only the latest submission before the deadline will be graded.
   - You may check the last submission status at any time:

   ```
   feedback 3311 Project
   ```

   - You may save the check result for later review:

   ```
   feedback 3311 Project > Project-check-result.txt
   ```

   Note. You will receive zero for submitting a project that cannot be compiled.

8.2 Phase 2 Submission

The submission procedure is identical to that for Phase 1 (i.e., just run the same `submit` command), except that items related to your project design document should be placed in the `docs` folder. More instructions will be updated (see Section 7).

9 Questions

There might be unclarity, typos, or even errors in this document. It is **your responsibility** to bring them up, early enough, for discussion. Questions related to the project requirements are expected to be posted on the on-line course forum. It is also **your responsibility** to frequently check the forum for any clarifications/changes on the project requirements.

10 Amendments

–