Comp160
Project 2
Fall 2018

The Project

For this project you'll scale up the space invaders game from lab 8 to include the following features:

- Multiple UFOs that land at constant rate, randomly move left or right, and randomly drop charges that fall at a constant rate (faster than UFO). UFOs blow up after being hit by one missile. We should be able to start the game with any number of UFOs.
- A single tank that is constantly moving left or right, wraps around the scene when it moves off one side or the other, can be hit by 2 charges but will blow up on the third hit, and can fire any number of missiles. Missiles should travel at a constant rate but should be faster than the UFO charges.
- The game tracks total number of UFOs shot down by the tank.
- The game ends when one of the UFOs land on the ground, when the tank is destroyed, or when all of the UFOs are shot down by the tank. When the game ends a game over scene should be displayed that reports if and how the user won or lost along with the final score.
- While the game is in progress the player should see their tank's hit points and their current score along with the UFOs and the tank.

The project will, once again, be organized into tiers that determine the overall level of your grade. Tiers are discussed below. Be sure you're clear on the expectations of each tier before you get into the work.

Project Tier 0 - Starter

You'll begin this project by defining a WorldState structure that can handle the sum total of the game. Early tiers will effectively ignore parts of the structure. The strategy here is to build a data type that can handle the end goal in order to avoid major rewrites[1] that come along with redefining your core worldstate data type.

[1] *refactoring*

Your tier 0 project begins with a well written and well documented data definitions for a world state that supports *all* the the game features described above. Included with definition should be several

constants that represent key game states[2]. You must then create a basic project starter a la lab 7. This includes: a main function with all the necessary game events represented, the signature, purpose, and header for all main event handler functions, and definitions for any essential game constants. Again, we're planning for a complete program but will proceed by designing and implementing a few pieces at a time.

[2] start, end, some combat, etc.

Project Tier 1

The tier 1 project adds all of the basic motion to the game but none of the combat features. This means UFOs that descend and move left or right at random[3] along with a movable tank that wraps around the scene as needed. There are no missiles, no charges, no score kept, and no tank hit points tracked. When a UFO lands the user should see the appropriate game over screen.

[3] See Exercise 99 in the text for pointers and discussion on random movement

Project Tier 2

The tier 2 project adds in the tank's ability to fire missiles. Score should now be kept and displayed. The game now ends if a UFO lands or if all of the UFOs are destroyed.

Project Tier 3

The third and final tier give the UFOs the ability to fight back. UFOs can now fire charges, the tank can take damage, tank hit-points should be displayed and the game will end if the tank runs out of hit points.

On Random Behaviors

BSL Racket provides a function called *random* that takes some natural number[4] n and produces a natural number from zero to $n - 1$ at random. Think of it as rolling an n sided die. Using *random(2)* gives you a coin flip that results in 0 or 1 where using *random(6)* is like rolling a standard six-sided die but with the results being zero to five, not one to six.

[4] Positive number or zero

This projects has two random behaviors in it: UFOs will randomly move left or right and they will randomly drop a charge. You can equate both of these to simple dice rolls. A good starting point for random movement would be *random(4)* where two possible values mean no sideways movement, one value means move left, and the final value means move right. This results in a UFO that stays put half the time and moves half the time. If you want to increase the rate of

movement, then reduce the number of options for not moving. Conversely, if you want the UFO to move to the sides less frequently you can increase the number of options that result in no movement. For example, doing the equivalent of rolling a six-sided die where four of the six results result in no sideways movement and the other two result in left and right movement respectively. Feel free to play around and find a movement rate that works for you. The movement itself can just be by some fixed amount[5]. Dropping charges can work the same way. Do the equivalent of a six-sided die roll with a 3 resulting in a dropped charge but all other outcomes meaning don't drop a charge. In this case you have a 1 in 6 chance of dropping a charge. This means a UFO that takes 20 ticks to get to the bottom would drop something in the ball park of 3 charges. Again, you can play with the proportions to change the rate at which charges drop.

[5] try something like 1/4 or 1/8 of the width of the UFO

When designing your randomized function be sure to stick to the two function design described in the book and the example you were given. In this case one function would, essentially, take the result of a die roll and produce the desired result and the other function would roll the die and hand the result off to the first function. For example, you could have a move-ufo function that takes the current UFO (a posn) and a number from [0,3] (the result of the roll) and produces the new UFO. You'd then write another function that takes only the UFO. It's definition might look something like this:

```
(define (move-ufo-random u)
  (move-ufo u (random 4)))
```

Notice it's job is really to roll the dice the hand the result off to another function which finishes out the process based on the result of the roll.

Grading

Your grade will be determined by two factors: your progress through the tiers determines the letter grade range that is open to you and the quality of your work will determine where you land within that range.

Tiers and Grade Ranges

Each tier corresponds to a range of grades as described below.

Tier	Grade Range
0	D
1	C
2	B
3	A

Programs that produce syntax errors when run can expect to receive a failing grade. Programs that run but crash often or lack any testing can expect receive something in the D range at best. No matter what, submit something that runs. Assuming your program successfully implements the features for a given tier, your grade will be determined by the quality of the work. For example, completing all the UFO and tank movement described in tier 1 could will get you something between a C- and a C+ depending on the quality of your work. Completing some of a tier can net your partial credit so long as your work is well done. For example, have all of the tier 1 features done, having a tank that fires missiles, but not dealing with score keeping or shooting down UFOs could, quality depending, receive as high as a B-.

Quality

A high quality program exhibits all the earmarks of intentional, systematic design and good programming style. Program data is appropriately defined. Functions are well documented. Incomplete functions are stubbed as opposed to commented out. Complete functions typically exhibit template structure when applicable. Complete and incomplete functions have a full set of tests. Signatures, purpose statements, tests and function definitions are all appropriately placed. Function and variable names are helpful and meaningful. Purpose statements are specific and concrete. Indentation of code follows expected Racket standards[6]. Lines are terminated to avoid print wrapping. Comments are used to aid the human reader. Whitespace is used to break up logical blocks of definitions. The degree to which you meet these standards, along with progress within the tier, determines where you fall within the grade range associated with your tier. It is entirely possible the poorly designed and styled code the completes three extensions can get a lower grade than a well designed and styled program that completes one or two extensions. Do not short yourself points by writing sloppy code.

[6] It's styled like the code in the book.

Important Dates

Date	What's Happening
Monday 11/26	Open Lab Time to For Project Work
Tuesday 11/27	Open Lab Time to For Project Work
Monday 12/3	Open Lab Time to For Project Work
Tuesday 12/4	Open Lab Time to For Project Work
Wednesday 12/5	Printed & Digital copy of code due by the end of the day.