Tower Defense: Extra Credit

1 Introduction

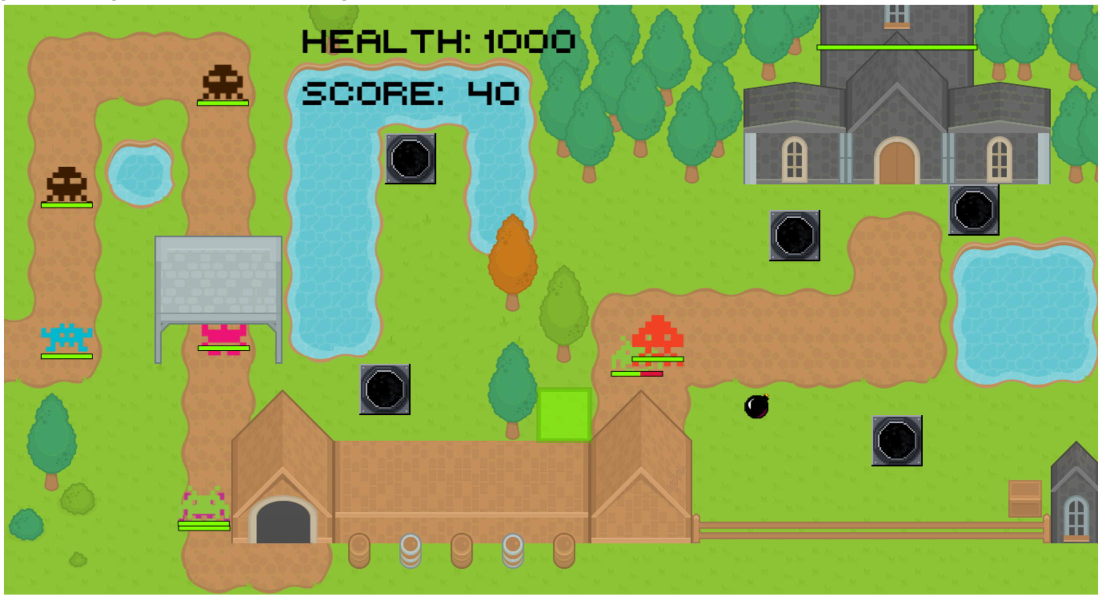
This extra credit homework is focused around a particular theme, but you don't have to get too into it if you don't want to. Grading will be entirely on a partial-credit basis, so you can do just a little bit to get some extra points without having to complete a full-fledged game.

2 Game

You're going to be writing a tower defense game. For those of you who haven't heard of it, see the attached video showing the sample game our amazing TA Michael has created. The game is a one player real-time strategy game, and the goal is to defend the home base from waves of oncoming enemies by placing turrets that fire at the enemies.

If the enemies reach the base, they begin damaging it until it is destroyed, at which point the game is over. Typically, the player is limited in how many turrets he or she may place at once, and has to destroy incoming enemies to earn more.

Your game might look something like:



...but of course can look entirely different. We've included the resources used in our implementation, but you're welcome to create or find your own.

3 Assignment/Grade Breakdown

This assignment is 100% open-ended. There will be no point deductions, only points given, as long as your code compiles. Below is a list of features you may implement. You can implement as many of them as you like to earn up to 100 points. Rough point estimates are given, but we reserve the right to change these values.

To be clear: this means that you could in theory get 100 points without writing a GUI, if you do all of the other functionality.

- **Game Model/Logic** - up to 50pts
  - Create a game framework with sensible classes/types for the underlying components.
  - Utilize good object-oriented principles as learned throughout the semester.
  - Make good use of interfaces or abstract classes,
  - Properly utilizes inheritance and polymorphism, etc.
- **Game Functionality** - up to 50pts, more for more features
  Each item worth roughly 5 points
  - Can place turrets
  - Enemies move across screen
  - Turrets shoot
  - Enemies die
  - Enemies deal damage to "home base" or some such structure that is being defended.
  - Visible health bars
  - Multiple waves of enemies
  - Additionally functionality encouraged. We will give more points at our discretion.
- **Graphics** - up to 20pts
  Even if you don't get the full game functionality working as mentioned above (or even if you do), you can still receive credit for having a GUI that does *something*. If you get all the functionality above working, you've probably already accomplished these. Points could be awarded for:
  - Animation
  - Multiple, distinct elements (enemies, towers, health/score labels, etc.)

- **MVC** - up to 10pts
  This only applies if you actually get both a backend and a GUI working, but you don't have to have all the game functionality working. If you can architect your game around MVC (or some other paradigm), you can receive points. Be sure to document this so we know to look for it.
- **App Structure & Packaging** - up to 25pts
  - Your code makes use of at least two packages that make sense (i.e, not contrived) and is arranged in a matching directory hierarchy
  - You include an ANT buildfile that can be used to compile and run your code
  - You include a runnable JAR file
- **Javadoc & Checkstyle** - 10pts
  - All code is thoroughly and meaningfully javadoc'd (i.e, don't put "This is the Tower class" as your description)
  - All code passes checkstyle with zero errors.
- **Misc.** - More points at our discretion
  - Use of some non-trivial algorithm for enemy AI. (Hint: lookup breadth first search, depth first search, etc.)
  - Uses at least three custom exceptions in a meaningful manner
  - Keeps track of levels
  - Allows saving/loading of game
  - Any other cool feature you want to add.

For example, you could write a fully functioning tower defense game with terrible code and terrible graphics, and you'd receive roughly 50pts. Or you could write no GUI or functional code, and instead just write all the game logic and models, javadoc and checkstyle your code, and package it up nicely, and receive roughly an 85.

4 `README.txt`

Because it's dead week and we're going to scramble to grade these, please include a file named `README.txt` that outlines everything for which you would like to receive credit. Anything not mentioned in the file will be ignored.

Again: **PLEASE BE SURE TO LIST ALL FEATURES IN THE README**

A sample readme may look like:

```
Game Model/Logic:
   have classes and interfaces, use inheritance/polymorphism, have game controller object
MVC:
   have models (enemy classes, turret classes), controller (TowerDefense), and view
       (TowerDefenseView)
Functionality:
   You can place turrets
   Enemies move across screen
   Enemies destroy turrets
   Turrets shoot back
   Enemies can destroy home base
AI:
   My enemies use the A$^*$ algorithm to find and destroy turrets before moving on to the home
       base
```

5 Turn-in Procedure

Submit all of the Java source files you modified and resources your program requires to run to T-Square. Do not submit any compiled bytecode (`.class` files) or the Checkstyle jar file. When you're ready, double-check that you have submitted and not just saved a draft.

Please remember to run your code through Checkstyle!

Verify the Success of Your Submission to T-Square
Practice safe submission! Verify that your HW files were truly submitted correctly, the upload was successful, and that the files compile and run. It is solely your responsibility to turn in your homework and practice this safe submission safeguard.

1. After uploading the files to T-Square you should receive an email from T-Square listing the names of the files that were uploaded and received. If you do not get the confirmation email almost immediately, something is wrong with your HW submission and/or your email. Even receiving the email does not guarantee that you turned in exactly what you intended.
2. After submitting the files to T-Square, return to the Assignment menu option and this homework. It should show the submitted files.
3. Download copies of your submitted files from the T-Square Assignment page placing them in a new folder.
4. Recompile and test those exact files.
5. This helps guard against a few things.
   (a) It helps insure that you turn in the correct files.
   (b) It helps you realize if you omit a file or files. [1] (If you do discover that you omitted a file, submit all of your files again, not just the missing one.)

[1]Missing files will not be given any credit, and non-compiling homework solutions will receive few to zero points. Also recall that late homework will not be accepted regardless of excuse. Treat the due date with respect. The real due date is midnight. Do not wait until the last minute!

(c) Helps find last minute causes of files not compiling and/or running.